SPEECH EMOTION DETECTION USING MACHINE LEARNING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING SCHOOL OF COMPUTING

Presented by:- S.Jagruthi

AURORA

INSTITUTE OF SCIENCE AND TECHNOLOGY (DEEMED TO BE UNIVERSITY)
Accredited with Grade "A" by NAAC

ABSTRACT

The speech signal is one of the most natural and fastest methods of communication between humans. Many systems have been developed by various researchers to identify the emotions from the speech signal. In differentiating between various emotions particularly speech features are more useful and if not clear is the reason that makes emotion recognition from speaker‘s speech very difficult. There are a number of the dataset available for speech emotions, it's modelling, and types that helps in knowing the type of speech. After feature extraction, another important part is the classification of speech emotions so the paper has compared and reviewed the different classifiers that are used to differentiate emotions such as sadness, neutral, happiness, surprise, anger, etc. The research also shows the improvement in emotion recognition system by making automatic emotion recognition system adding a deep neural network. The analysis has also been performed using different ML techniques for Speech emotions recognition accuracy in different languages.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 OUTLINE

One of the fastest and natural methods of communication between humans is a speech signal. For interaction between human and machine use of speech signal is the fastest and most efficient method. To maximum awareness of received message, all available senses are used by human‘s natural ability. For machine emotional detection is a very difficult task, on the other hand, it is natural for humans. So, knowledge related to emotion is used by an emotion recognition system in such a way that there is an improvement in communication between machine and human. The female or make speakers emotions find out through speech in speech emotion recognition. These features make a base for speech processing. In differentiating between various emotions particularly speech features are more useful is not clear is the reason that makes emotion recognition from speakers‘ speech very difficult. There is an introduction of accosting variability due to the existence of different speaking rates, styles, sentences and speakers that affects the features of speech. Different emotions may be shown by the same utterance and there are different portions of the spoken utterance of each correspond emotion that makes it difficult to differentiate these portions of utterance. The emotion expression depends on the culture and environment of the speaker that creates another problem as there is variation in the style of speaking by the variations in environment and culture. Transient and long terms emotion are two types of emotions and it is not clear about the type of emotion detected by recognizer. Speech information recognized emotions may be speaker independent or speaker dependent.

1.2 DATAOVERVIEW

Figure 1 – RAVDESS dataset.

This dataset is a combination of song files and speech files. we have a total of 24 actor files in both speech and song[13]. In 40 audio files in song files. The emotions are labelled as follows: 01-'neutral', 02-'calm', 03-'happy', 04-'sad', 05-'angry', 06- 'fearful', 07-'disgust', 08 -'surprised'.

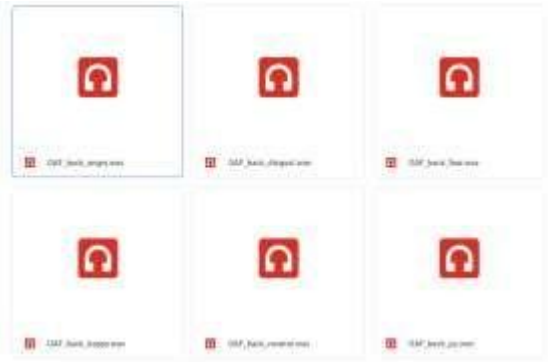

Figure2 – TESS dataset.
This dataset contains a total of 2800 audio file.

CHAPTER 2

LITERATURE REVIEW

An i-vector GPLDA System for Speech based Emotion Recognition

In this paper, we propose the use of a Gaussian Probabilistic Linear Discriminant Analysis back-end for utterance level emotion classification based on i-vectors representing the distribution of frame level MFCC features. Experimental results based on the IEMOCAP corpus show that the GPLDA back-end outperforms an SVM based back-end while being less sensitive to i-vector dimensionality, making the proposed framework more robust to parameter tuning during system development.

Reconstruction-error-based learning for continuous emotion recognition in speech

To advance the performance of continuous emotion recognition from speech, we introduce a reconstruction-error-based learning framework with memory-enhanced Recurrent Neural Networks. In the framework, two successive RNN models are adopted, where the first model is used as an auto encoder for reconstructing the original features, and the second is employed to perform emotion prediction. The RE of the original features is used as a complementary descriptor, which is merged with the original features and fed to the second model. The assumption of this framework is that the system has the ability to learn its _drawback' which is expressed by the RE. Experimental results on the RECOLA database show that the proposed framework significantly outperforms the baseline systems without any RE information in terms of Concordance Correlation Coefficient.

Speech-based Emotion Recognition and Next Reaction Prediction

Communication through voice is one of the main components of affective computing in human-computer interaction. In this type of interaction, properly comprehending the meanings of the words or the linguistic category and recognizing the emotion included in the speech is essential for enhancing the performance. In order to model the emotional state, the speech waves are utilized, which bear signals standing for emotions such as boredom, fear, joy and sadness. In the first step of the emotional reaction prediction system proposed in this

paper,
different emotions are recognized by means of different types of classifiers. The second step is the prediction of a sequence of the next emotional reactions using neural networks. The sequence is extracted based on the speech signals being digitized at tenths of a second, after concatenating the different speech signals ofeach subject. The prediction problem is solved as a nonlinear auto-regression time-series neural network with the assumption that the variables are defined as data-feedback time-series. The best average recognition rate is 86.25%, which is achieved by the Random Forest classifier, and the average prediction rate of reactions by using neural networks.

Speech Based Emotion Recognition Using Spectral Feature Extraction and an Ensemble of kNN Classifiers

Security (and cyber security) is an important issue in existing and developing technology. It is imperative that cyber security go beyond password based systems to avoid criminal activities. A human biometric and emotion based recognition framework implemented in parallel can enable applications to access personal or public information securely. The focus of this paper is on the study of speech based emotion recognition using a pattern recognition paradigm with spectral feature extraction and an ensemble of *k* nearest neighbor classifiers. The five spectral features are the linear predictive cepstrum, mel frequency cepstrum, line spectral frequencies, adaptive component weighted cepstrum and the post filter cepstrum. The bagging algorithm is used to train the ensemble of kNNs. Fusion is implicitly accomplished by ensemble classification. The LDC emotional prosody speech Database is used in all the experiments. Results show that the maximum gain in performance is achieved by using two kNNs as opposed to using a single kNN.

Towards Robust Speech-Based Emotion Recognition

Abstract-Maintaining the robustness of a speech processing system in the presence of noise is a challenge. This paper shows frequency subband architecture can improve robustness of an emotion recognition system when signals are corrupted by selective noise. This paper also demonstrates that feature
Selection based on some mutual-information criterion can give us the most effective subset of features to get a better result.

CHAPTER 3
METHODOLOGY

PURPOSE

The purpose of this document is speech emotion detection using machine learning algorithms. In detail, this document will provide a general description of our project, including user requirements, product perspective, and overview of requirements, general constraints. In addition, it will also provide the specific requirements and functionality needed for this project - such as interface, functional requirements and performance requirements.

SCOPE

The scope of this SRSdocument persists for the entire life cycle of the project. This document defines the final state of the software requirements agreed upon by the customers and designers. Finally at the end of the project execution all the functionalities may be traceable from the SRSto the product. The document describes the functionality, performance, constraints, interface and reliability for the entire life cycle of the project.

EXISTING SYSTEM

The speech emotion detection system is implemented as a Machine Learning (ML) model. The steps of implementation are comparable to any other ML project, with additional fine-tuning procedures to make the model function better. The flowchart represents a pictorial overview of the process (see Figure 1). The first step is data collection, which is of prime importance. The model being developed will learn from the data provided to it and all the decisions and results that a developed model will produce is guided by the data. The second step, called feature engineering, is a collection of several machine learning tasks that are executed over the collected data. These procedures address the several data representation and data quality issues. The third step is often considered the core of an ML project where an algorithmic based model is developed. This model uses an ML algorithm to learn about the data and train itself to respond to any new data it is exposed to. The final step is to evaluate the functioning of the built model. Very often, developers repeat the steps of developing a model and evaluating it to compare the performance of different algorithms. Comparison results help to choose the appropriate ML algorithm most relevant to the problem.

PROPOSED SYSTEM

In this current study, we presented an automatic speech emotion

recognition (SER) system using machine learning algorithms to classify the emotions.The performance of the emotion detection system can greatly influence the overall performance of the application in many ways and can provide many advantages over the functionalities of these applications. This research presents a speech emotion detection system with improvements over an existing system in terms of data, feature selection, and methodology that aims at classifying speech percepts based on emotions, more accurately.

ALGORITHMS USED

CLASSIFIERS

Classification is defined as the process of recognition, understanding, and grouping of objects and ideas into preset categories a.k.a ¯sub-populations.‖ With the help of these pre-categorized training datasets, classification in machine learning programs leverage a wide range of algorithms to classify future datasets into respective and relevant categories. Classification algorithms used in machine learning utilize input training data for the purpose of predicting the likelihood or probability that the data that follows will fall into one of the predetermined categories. One of the most common applications of classification is for filtering emails into ¯spam‖ or ¯non-spam‖, as used by today‘s top email service providers.In short, classification is a form of ¯pattern recognition,‖. Here, classification algorithms applied to the training data find the same pattern (similar number sequences, words or sentiments, and the like) in future data sets.

We will explore classification algorithms in detail, and discover how a text analysis software can perform actions like sentiment analysis - used for categorizing unstructured text by opinion polarity (positive, negative, neutral, and the like).

SVC

SVM algorithms classify data and train models within super finite degrees of polarity,creating a 3-dimensional classification model that goes beyond just X/Y predictive axes. Take a look at this visual representation to understand how SVM algorithms work. We have two tags: red and blue, with two data features: X and Y, and we train our classifier to output an X/Y coordinate as either red or blue.

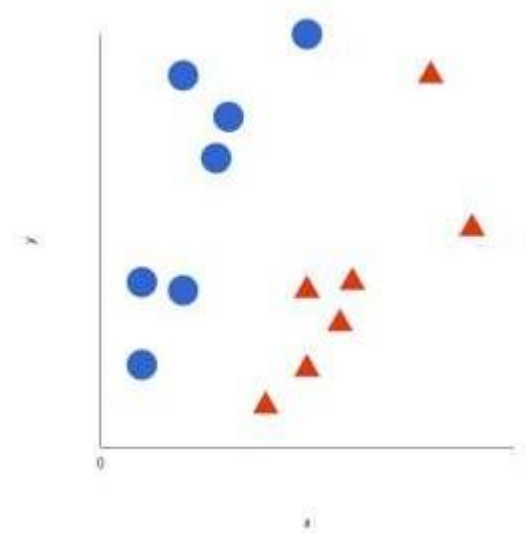

Figure 3 – SVC Without Hyperplane

The SVM assigns a hyperplane that best separates (distinguishes between) the tags. In two dimensions this is simply a straight line. Blue tags fall on one side of the hyperplane and red on the other. In sentiment analysis these tags would be Positive and Negative.

To maximize machine learning model training, the best hyperplane is the one with the largest distance between each tag:

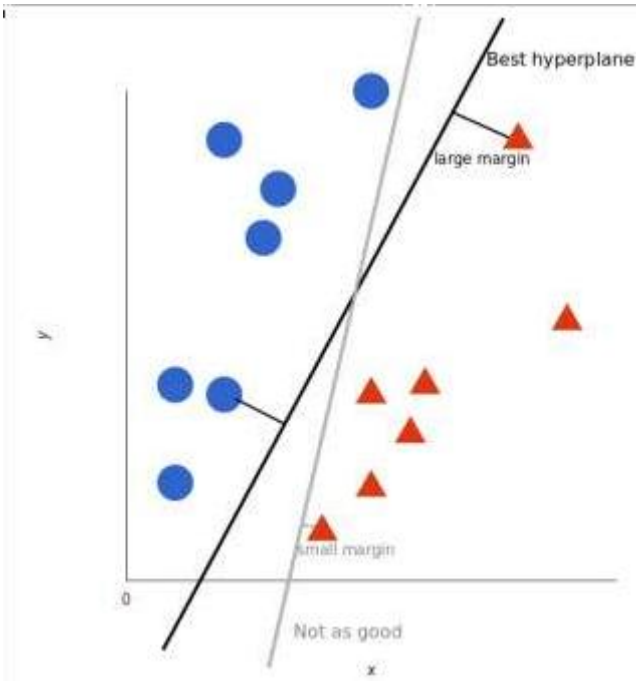

Figure 4 – SVC WithHyperplane

SVM algorithms make excellent classifiers because, the more complex the data, the more accurate the prediction will be. Imagine the above as a 3-dimensional output, with a Z-axis added, so it becomes a circle.

Mapped back to 2D, with the best hyperplane, it looks like this:

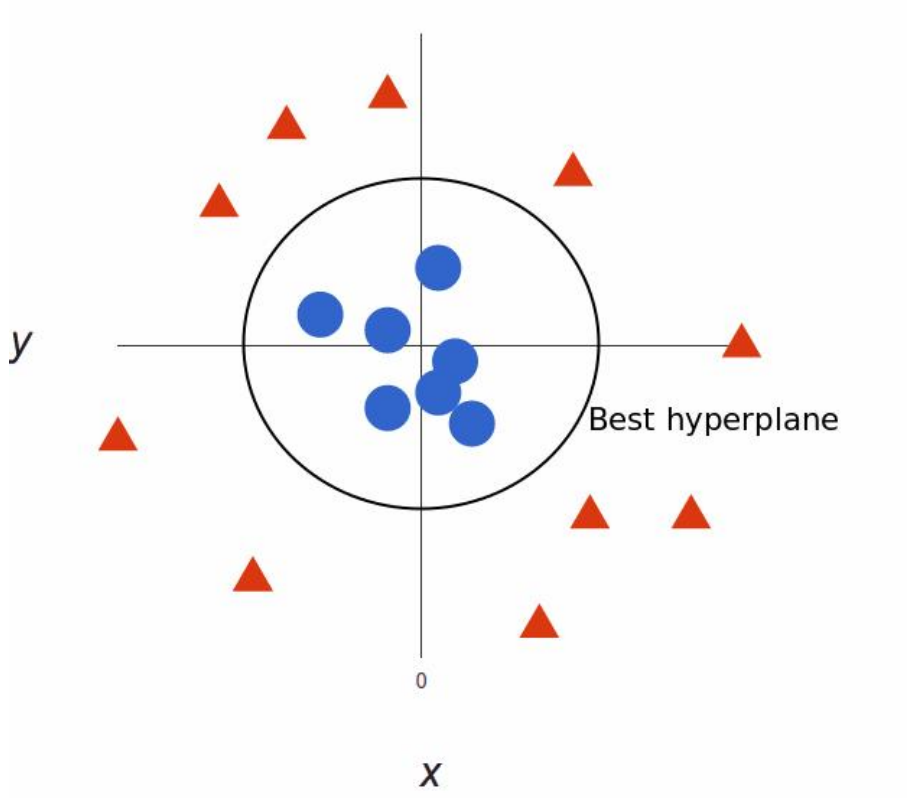

Figure 5 – SVC Best Hyperplane

SVM algorithms create super accurate machine learning models because they‘re multidimensional.

RANDOM FOREST CLASSIFIER

The term ¯Random Forest Classifier‖ refers to the classification algorithm made up of several decision trees. The algorithm uses randomness to build each individual tree to promote uncorrelated forests, which then uses the forest‘s predictive powers to make accurate decisions.

Random forestclassifiers fall under the broad umbrella of ensemble-based learning methods. They are simple to implement, fast in operation, and have proven to be extremely successful in a variety of domains. The key principle underlying the random forest approach comprises the construction of many ¯simple‖ decision trees in the training stage and the majority vote (mode) across them in the classification stage. Among other benefits, this voting strategy has the effect of correcting for the undesirable property of decision trees to overfit training data. In the training stage, random forests apply the general technique known as bagging to individual trees in the ensemble. Bagging repeatedly selects a random sample with replacement from the

training set and fits trees to these samples. Each tree is grown without any pruning. The number of trees in the ensemble is a free parameter which is readily learned automatically using the so-called out-of-bag error .
Much like in the case of naïve Bayes- and k-nearest neighbor-based algorithms, random forests are popular in part due to their simplicity on the one hand, and generally good performance on the other. However, unlike the former two approaches, random forests exhibit a degree of unpredictability as regards the structure of the final trained model. This is an inherent consequence of the stochastic nature of tree building. As we will explore in more detail shortly, one of the key reasons why this characteristic of random forests can be a problem in regulatory reasons—clinical adoption often demands a high degree of repeatability not only in terms of the ultimate performance of an algorithm but also in terms of the mechanics as to how a specific decision is made.

KNeighborsClassifier

The concept of the k-nearest neighbor classifier can hardly be simpler described. This is an old saying, which can be found in many languages and many cultures.This means that the concept of the k-nearest neighbor classifier is part of our everyday life and judging: Imagine you meet a group of people, they are all very young, stylish and sportive. They talk about there friend Ben, who isn't with them. So, what is your imagination of Ben? Right, you imagine him as being yong, stylish and sportive as well.If you learn that Ben lives in a neighborhood where people vote conservative and that the average income is above 200000 dollars a year? Both his neighbors make even more than 300,000 dollars per year? What do you think of Ben? Most probably, you do not consider him to be an underdog and you may suspect him to be a conservative as well?

The principle behind nearest neighbor classification consists in finding a predefined number, i.e. the 'k' - of training samples closest in distance to a new sample, which has to be classified. The label of the new sample will be defined from these neighbors. k-nearest neighbor classifiers have a fixed user defined constant for the number of neighbors which have to be determined. There are also radius-based neighbor learning algorithms, which have a varying number of neighbors based on the local density of points, all the samples inside of a fixed radius. The distance can, in general, be any metric measure: standard Euclidean distance is the most common choice. Neighbors-based methods are known as non-generalizing machine learning methods, since they simply "remember" all of its training data. Classification can be computed by a majority vote of the nearest neighbors of the unknown sample.

The k-NN algorithm is among the simplest of all machine learning algorithms, but despite its simplicity, it has been quite successful in a large number of classification and regression problems, for example character recognition or image analysis.

The algorithm for the k-nearest neighbor classifier is among the simplest of all machine learning algorithms. k-NN is a type of instance-based learning, or lazy learning. In machine learning, lazy learning is understood to be a learning method in which generalization of the training data is delayed until a query is made to the system. On the other hand, we have eager learning, where the system usually generalizes the training data before receiving queries. In other words: The function is only approximated locally and all the computations are performed, when the actual classification is being performed.

MLP CLASSIFIER

MLPClassifier stands for Multi-layer Perceptron classifier which in the name itself connects to a Neural Network. Unlike other classification algorithms such as Support Vectors or Naive Bayes Classifier, MLPClassifier relies on an underlying Neural Network to perform the task of classification.

A multi-layer rather than a single layer network is required since a single layer perceptron (SLP) can only compute a linear decision boundary, which is not flexible enough for most realistic learning problems. For a problem that is linearly separable, (that is capable of being perfectly separated by linear decision boundary), the perceptron convergence theorem guarantees convergence. In its simplest form, SLP training is based on the simple idea of adding or subtracting a pattern from the current weights when the target and predicted class disagrees, otherwise the weights are unchanged. For a non-linearly separable problem, this simple algorithm can go on cycling indefinitely. The modification known as least mean square (LMS) algorithm uses a mean squared error cost function to overcome this difficulty, but since there is only a single perceptron, the decision boundary is still linear. An MLP is a universal approximator [6] that typically uses the same squared error function as LMS. However, the main difficulty with the MLP is that the learning algorithm has a complex error surface, which can become stuck in local minima. There does not exist any MLP learning algorithm that is guaranteed to converge, as with SLP. The popular MLP back-propagation algorithm has two phases, the first being a forward pass, which is a forward simulation for the current training pattern and enables the error to be calculated. It is followed by a backward pass, that

calculates for each weight in the network how a small change will affect the error function. The derivative calculation is based on the application of the chain rule, and training typically proceeds by changing the weights proportional to the derivative.

Decision tree classifier

The decision tree acquires knowledge in the form of a tree, which can also be rewritten as a set of discrete rules to make it easier to understand. The main advantage of the decision tree classifier is its ability to using different feature subsets and decision rules at different stages of classification. As shown in Figure 4.6, a general decision tree consists of one root node, a number of internal and leaf nodes, and branches. Leaf nodes indicate the class to be assigned to a sample. Each internal node of a tree corresponds to a feature, and branches represent conjunctions of features that lead to those classifications. For food quality evaluation using computer vision, the decision tree has been applied to the problem of meat quality grading (Song et al., 2002) and the classification of ¯in the shell‖ pistachio nuts (Ghazanfari et al., 1998).

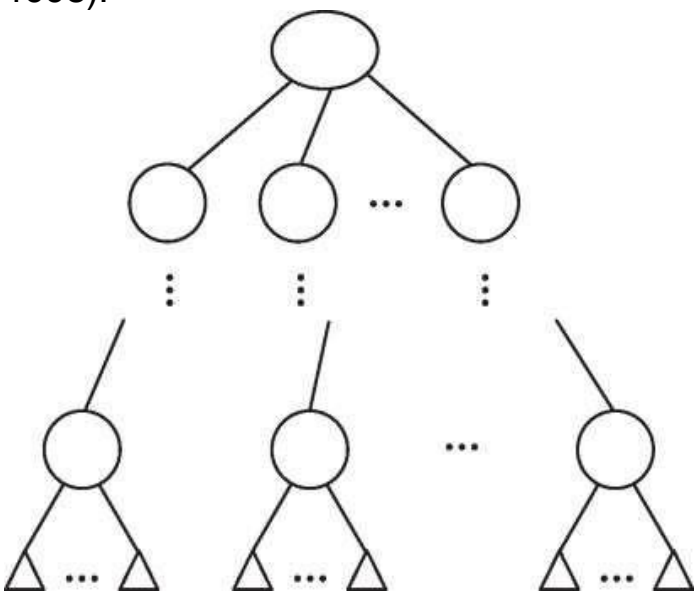

Figure 6 -Decision tree classifier

The performance of a decision tree classifier depends on how well the tree is constructed from the training data. A decision tree normally starts from a root node, and proceeds to split the source set into subsets, based on a feature value, to generate subtrees. This process is repeated on each derived subset in a recursive manner until leaf nodes are created.

REGRESSORS

A statistical approach that estimates the relationships among variables and predicts future outcomes or items in a continuous data set by solving for the pattern of past inputs, such as linear regression in statistics. Regression is foundational to machine learning and artificial intelligence. It predicts a real numbered label given an unlabeled example.

Regression analysis is the process of estimating the relationship between a dependent variable and independent variables. In simpler words, it means fitting a function from a selected family of functions to the sampled data under some error function. Regression analysis is one of the most basic tools in the area of machine learning used for prediction. Using regression you fit a function on the available data and try to predict the outcome for the future or hold-out datapoints. This fitting of function serves two purposes.

You can estimate missing data within your data range (Interpolation)

You can estimate future data outside your data range (Extrapolation)

Some real-world examples for regression analysis include predicting the price of a house given house features, predicting the impact of SAT/GRE scores on college admissions, predicting the sales based on input parameters, predicting the weather, etc.

Support Vector Regression

Support Vector Regression is a supervised learning algorithm that is used to predict discrete values. Support Vector Regression uses the same principle as the SVMs. The basic idea behind SVR is to find the best fit line. In SVR, the best fit line is the hyperplane that has the maximum number of points.

Unlike other Regression models that try to minimize the error between the real and predicted value, the SVR tries to fit the best line within a threshold value. The threshold value is the distance between the hyperplane and boundary line. The fit time complexity of SVR is more than quadratic with the number of samples which makes it hard to scale to datasets with more than a couple of 10000 samples.

For large datasets, Linear SVR or SGD Regressor is used. Linear SVR provides a faster implementation than SVR but only considers the linear kernel. The model produced by Support Vector Regression depends only on a subset of the training data, because the cost function ignores samples whose prediction is close to their target.

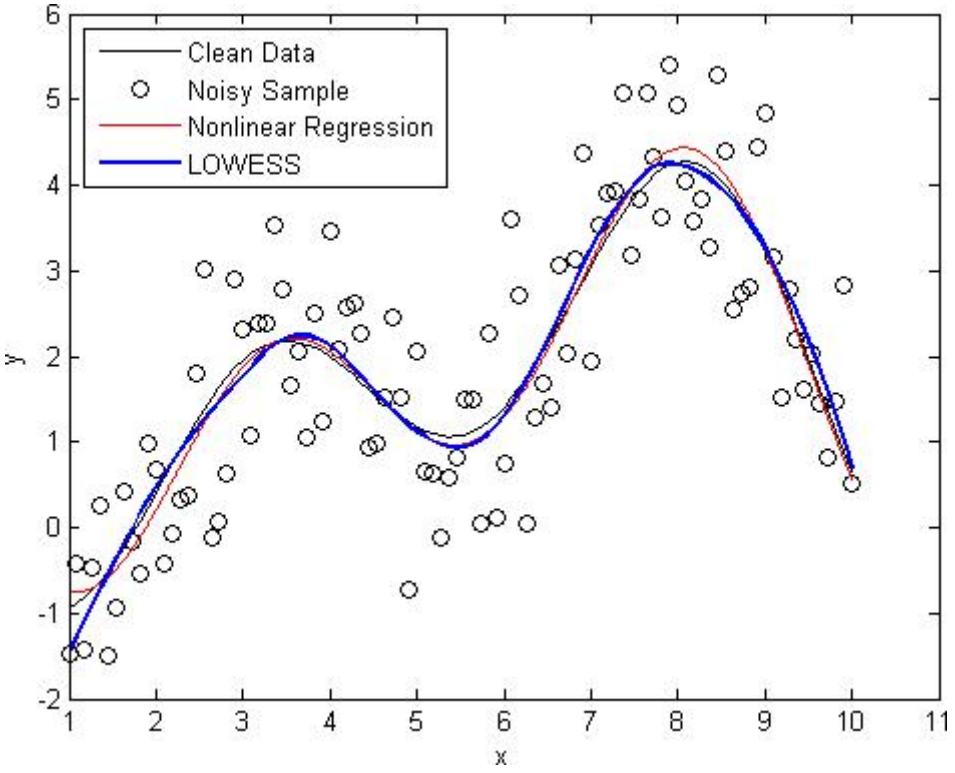

Figure 7 -Support Vector Regression

Random Forest Regression

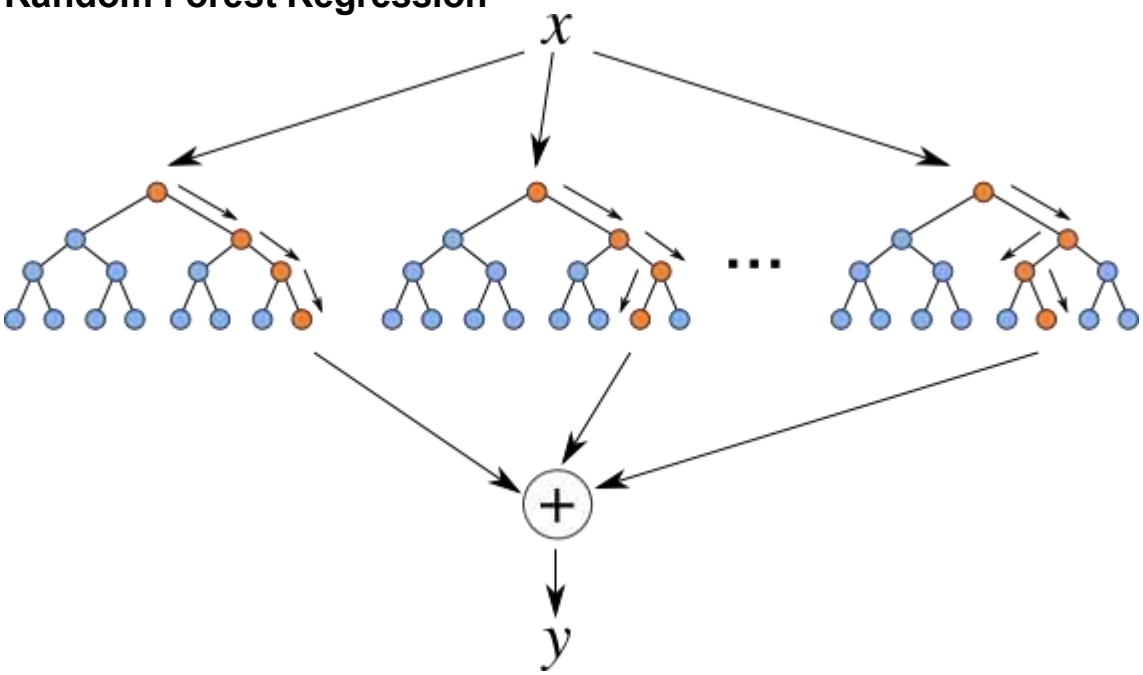

Figure 8 – Procedure of random forest

Random Forest Regression is a supervised learning algorithm that uses ensemble learning method for regression. Ensemble learning method is a technique that combines predictions from multiple machine learning algorithms to make a more accurate prediction than a single model.

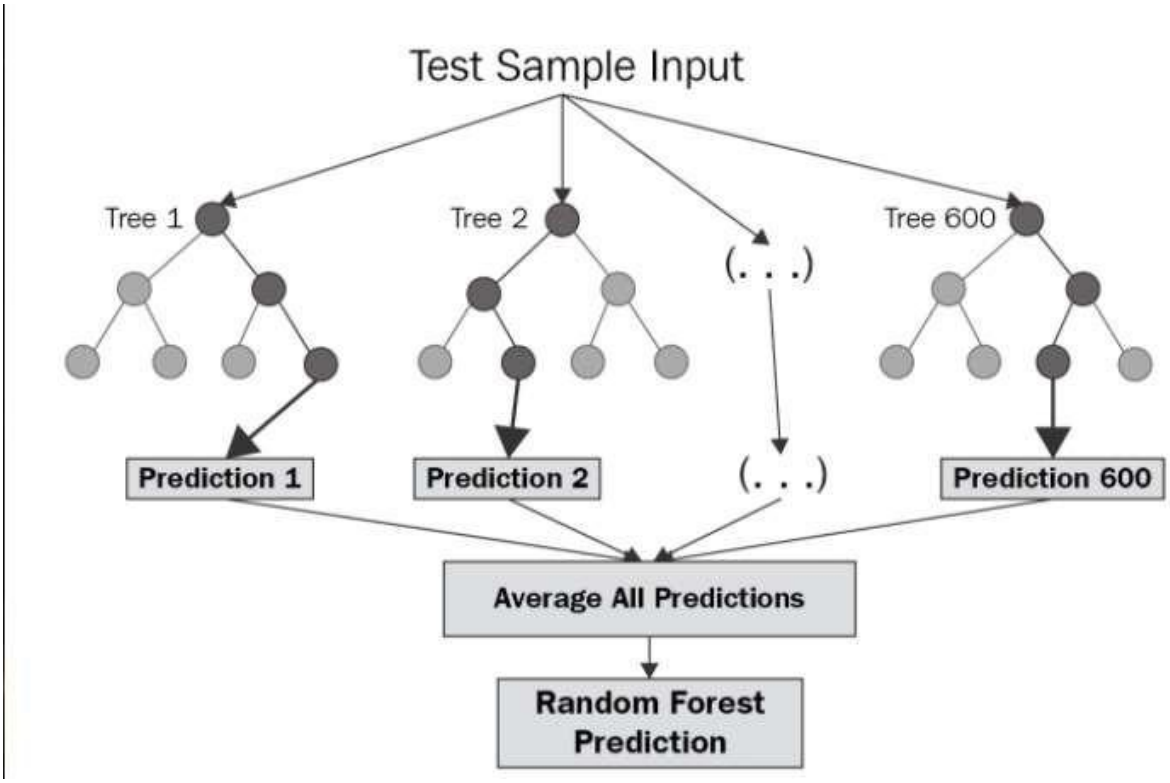

Figure 9 – structure of a Random Forest

The diagram above shows the structure of a Random Forest. You can notice that the trees run in parallel with no interaction amongst them. A Random Forest operates by constructing several decision trees during training time and outputting the mean of the classes as the prediction of all the trees. To get a better understanding of the Random Forest algorithm, let's walk through the steps:
Pick at random *k* data points from the training set.
Build a decision tree associated to these *k* data points.
Choose the number *N* of trees you want to build and repeat steps 1 and 2.
For a new data point, make each one of your *N*-tree trees predict the value of *y* for the data point in question and assign the new data point to the average across all of the predicted *y* values.
A Random Forest Regression model is powerful and accurate. It usually performs great on many problems, including features with non-linear relationships. Disadvantages, however, include the following: there is no interpretability, overfitting may easily occur, we must choose the number of trees to include in the model.

K-Nearest Neighbours Regression
KNN regression is a non-parametric method that, in an intuitive manner, approximates the association between independent variables and the continuous outcome by averaging the observations in the

same *neighbourhood*. The size of the neighbourhood needs to be set by the analyst or can be chosen using cross-validation (we will see this later) to select the size that minimises the mean-squared error.
While the method is quite appealing, it quickly becomes impractical when the dimension increases, i.e., when there are many independent variables.

The trade off with higher values of k is the loss of granularity in the decision boundary. Setting k very high will tend to give you smooth boundaries, but the real world boundaries you‘re trying to model might not be perfectly smooth.
Practically speaking, we can use the same sort of nearest neighbors approach for regressions, where we want an individual value rather than a classification. Consider the following regression below:

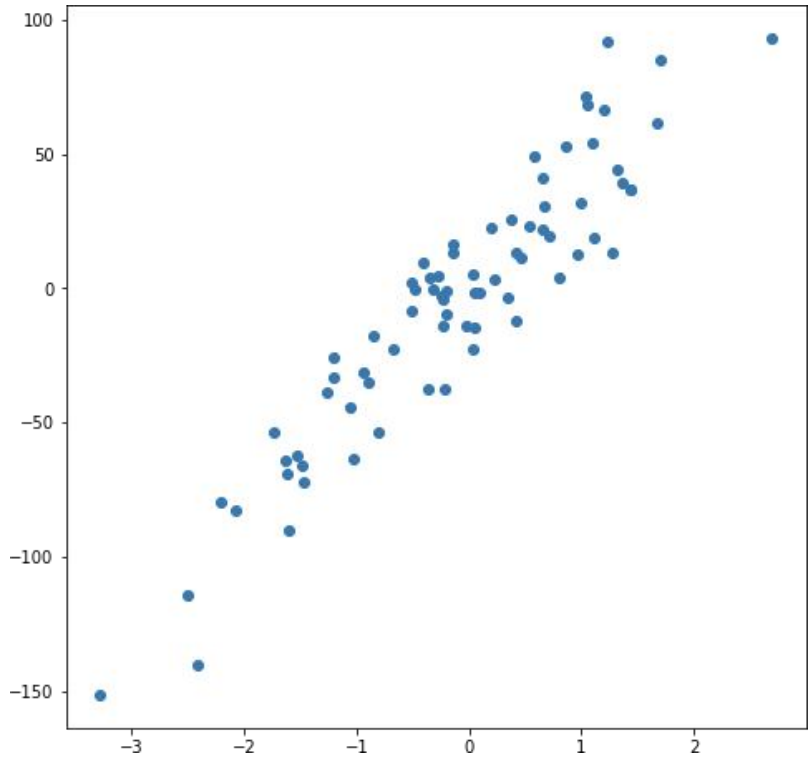

Figure 10 – A Simple regression example
A simple regression example
The data was randomly generated, but was generated to be linear, so a linear regression model would naturally fit this data well. I want to point out, though, that you can approximate the results of the linear method in a conceptually simpler way with a K-nearest neighbors approach. Our _regression‘ in this case won‘t be a single formula like an OLS model would give us, but rather a best predicted output value for any given input. Consider the value of -.75 on the x-axis, which I‘ve marked with a vertical line without solving any equations we can come to a reasonable approximation of what the output should be just by considering the

nearby points.Three points near a chosen x-value
It makes sense that the predicted value should be near these points, not much lower or higher. Perhaps a good prediction would be the average of these points:

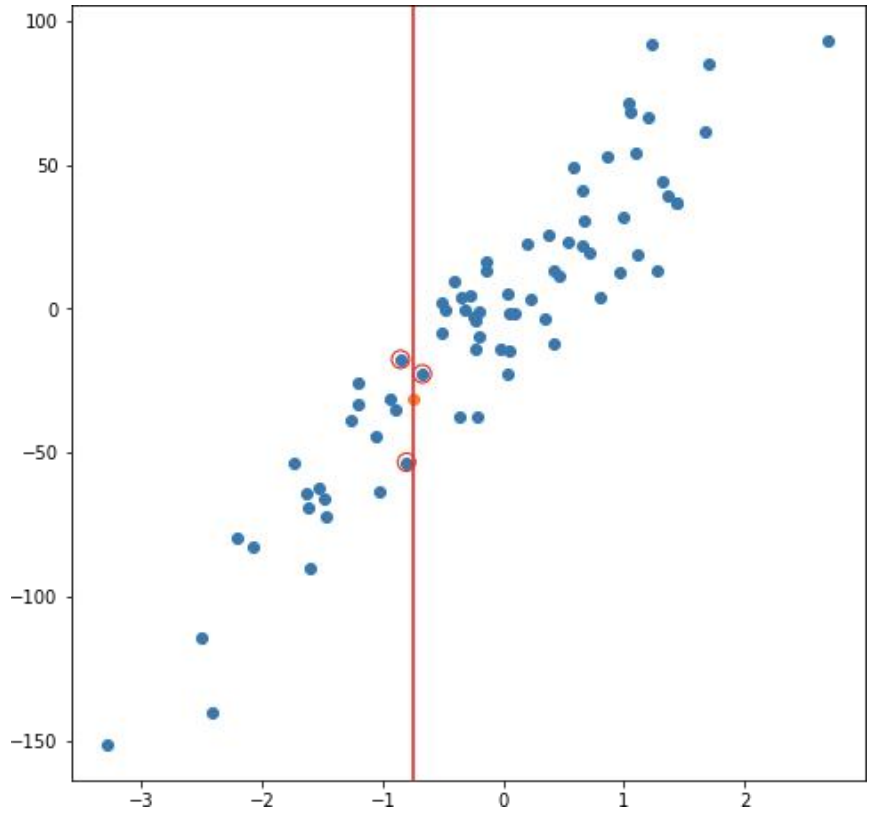

Figure 11 – An Average of outputs

You can imagine doing this for all the possible input values and coming up with predictions everywhere:

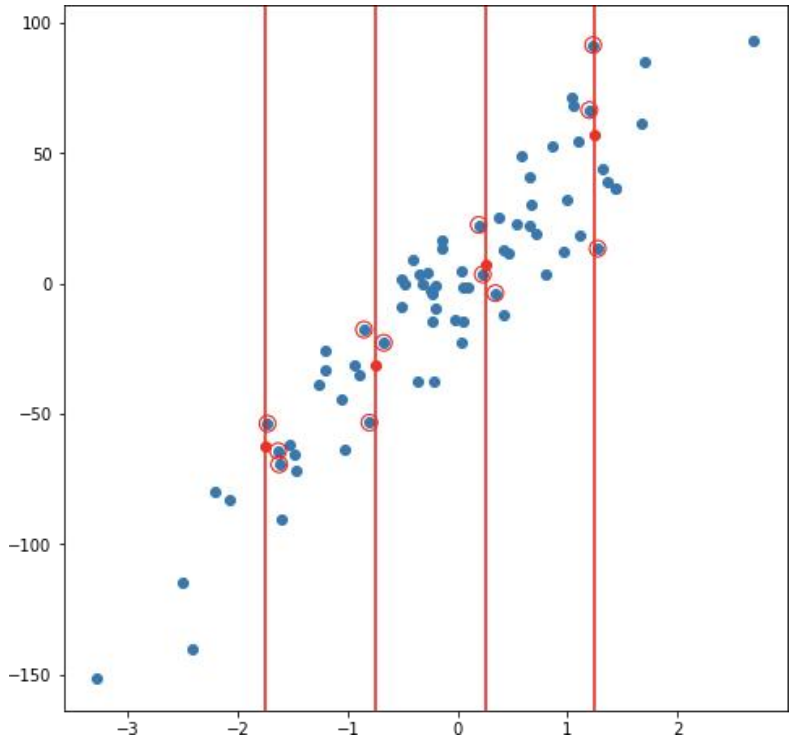

Figure 12–Predictions for different values

Connecting all these predictions with a line gives us our regression:

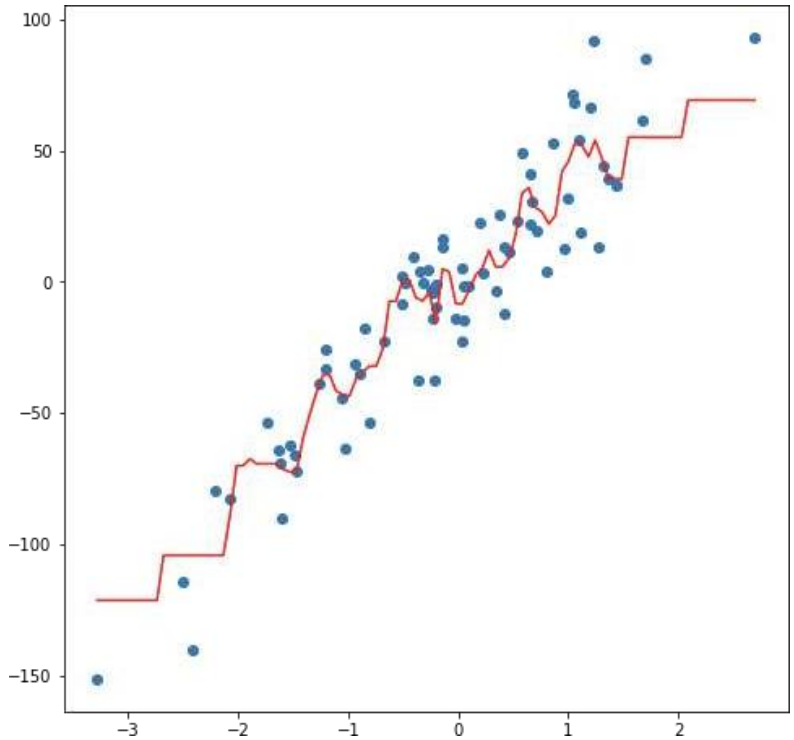

Figure 13 – KNN Regression

MLP Regressor

MLPRegressor implements a multi-layer perceptron (MLP) that trains using backpropagation with no activation function in the output layer, which can also be seen as using the identity function as activation

function. Therefore, it uses the square error as the loss function, and the output is a set of continuous values. When it comes to regression, there is a little more to say about the MLP. As it turns out, the only thing that changes is the activation function for the final nodes in the network that produces predictions. They allow for a wide range of outputs, not just the output from a set of classes. All the issues and hyperparameters are the same, as in the case of classification. Of course, in the regression context, you may end up making different choices than for classification. As well as MLPClassifier, there is its regressor sibling,MLPRegressor. The two share an almost identical interface.The main difference between the two is the loss functions used by each of them and the activation functions of the output layer. The regressor optimizes a squared loss, and the last layer is activated by an identity function. All other hyperparameters are the same, including the four activation options for the hidden layers.Both estimators have a partial_fit()method. You can use it to update the model once you get a hold of additional training data after the estimator has already been fitted.score()in MLPRegressorcalculates the regressor'sR2, as opposed to the classifier's accuracy, which is calculated byMLPClassifier.

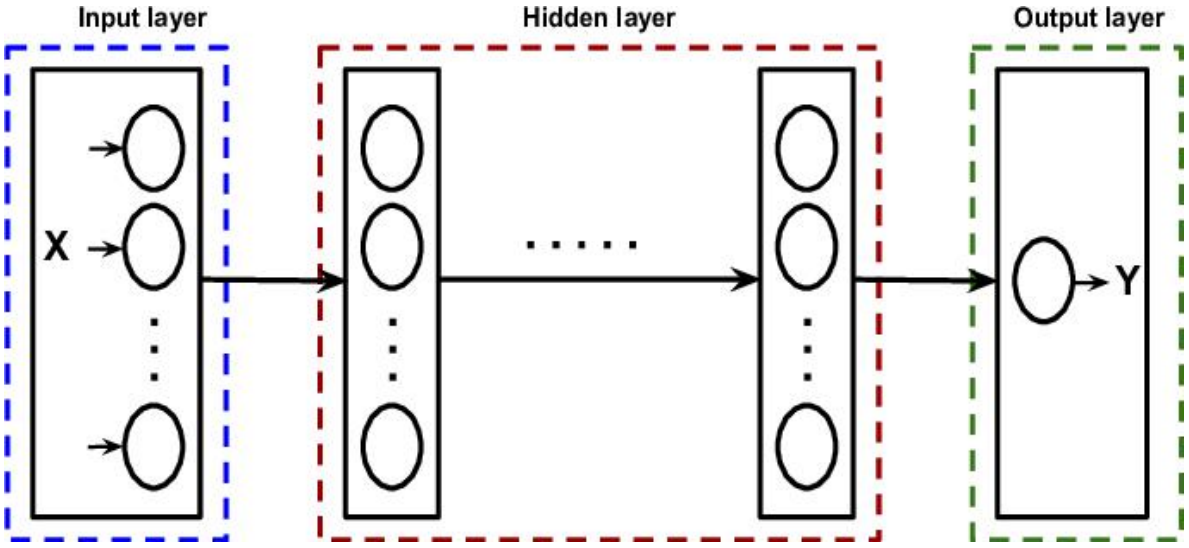

Figure 14 -MLP Regression

Decision Tree Regression:

Decision tree regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output. Continuous output means that the output/result is not discrete, i.e., it is not represented just by a discrete, known set of numbers or values. A 1D regression with decision

tree.The decision trees is used to fit a sine curve with addition noisy observation. As a result, it learns local linear regressions approximating the sine curve.We can see that if the maximum depth of the tree (controlled by the max_depth parameter) is set too high, the decision trees learn too fine details of the training data and learn from the noise, i.e. they overfit.

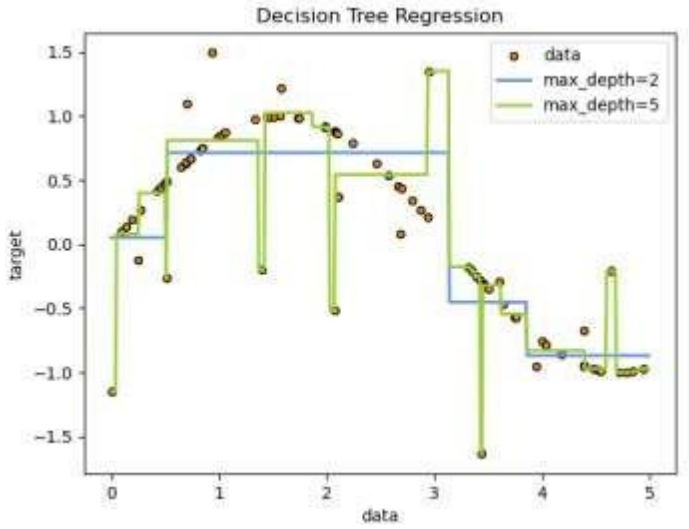

Figure 15 – Decision Tree Regression

SYSTEM DESIGN

INPUT DESIGN

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

What data should be given as input?

How the data should be arranged or coded?

The dialog to guide the operating personnel in providing input.

Methods for preparing input validations and steps to follow when error occur.

OUTPUT DESIGN

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system‘s relationship to help user decision-making.
The output form of an information system should accomplish one or more of the following objectives.
Convey information about past activities, current status or projections of the
Future.
Signal important events, opportunities, problems, or warnings.
Trigger an action.
Confirm an action

DATA FLOW DIAGRAM

The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.
The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.
DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.
DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.

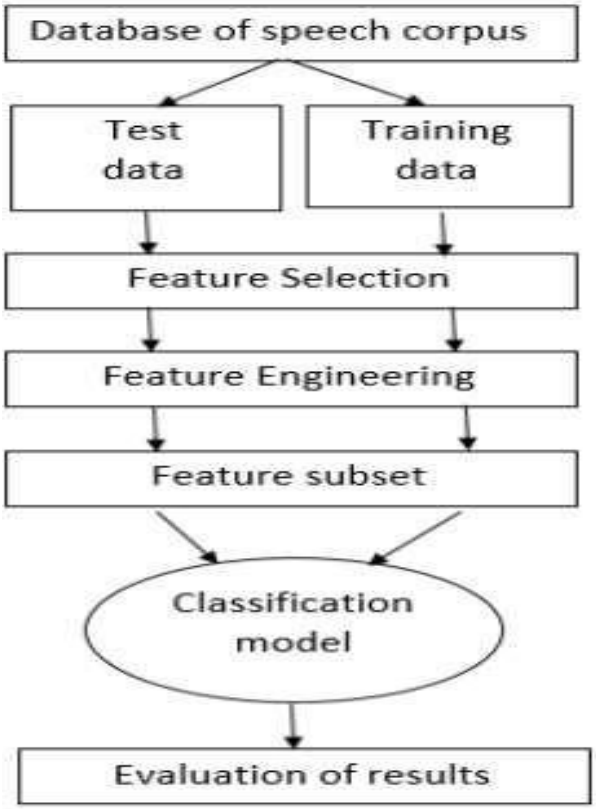

Figure 16 – Data Flow Diagram

MODULES

- Speech input Module
- Feature extraction and selection
- Classification
- Recognized emotional output

MODULE DESCRIPTION

1 Speech input Module

Input to the system is speech taken with the help of audio. Then equivalent digital representation of received audio is produced through sound file.

2 Feature extraction and selection

There are so many emotional states of emotion and emotion relevance is used to select the extracted speech features. For speech feature

extraction to selection corresponding to emotions all procedure revolves around the speech signal.

3 Classification Module

Finding a set of significant emotions for classification is the main concern in speech emotion recognition system. There are various emotional states contains in a typical set of emotions that makes classification a complicated task.

4 Recognized emotional output

Fear, surprise, anger, joy, disgust and sadness are primary emotions and naturalness of database level is the basis for speech emotion recognition system evaluation.

SYSTEM ARCHITECTURE

Describing the overall features of the software is concerned with defining the requirements and establishing the high level of the system. During architectural design, the various web pages and their interconnections are identified and designed. The major software components are identified and decomposed into processing modules and conceptual data structures and the interconnections among the modules are identified. The following modules are identified in the proposed system.

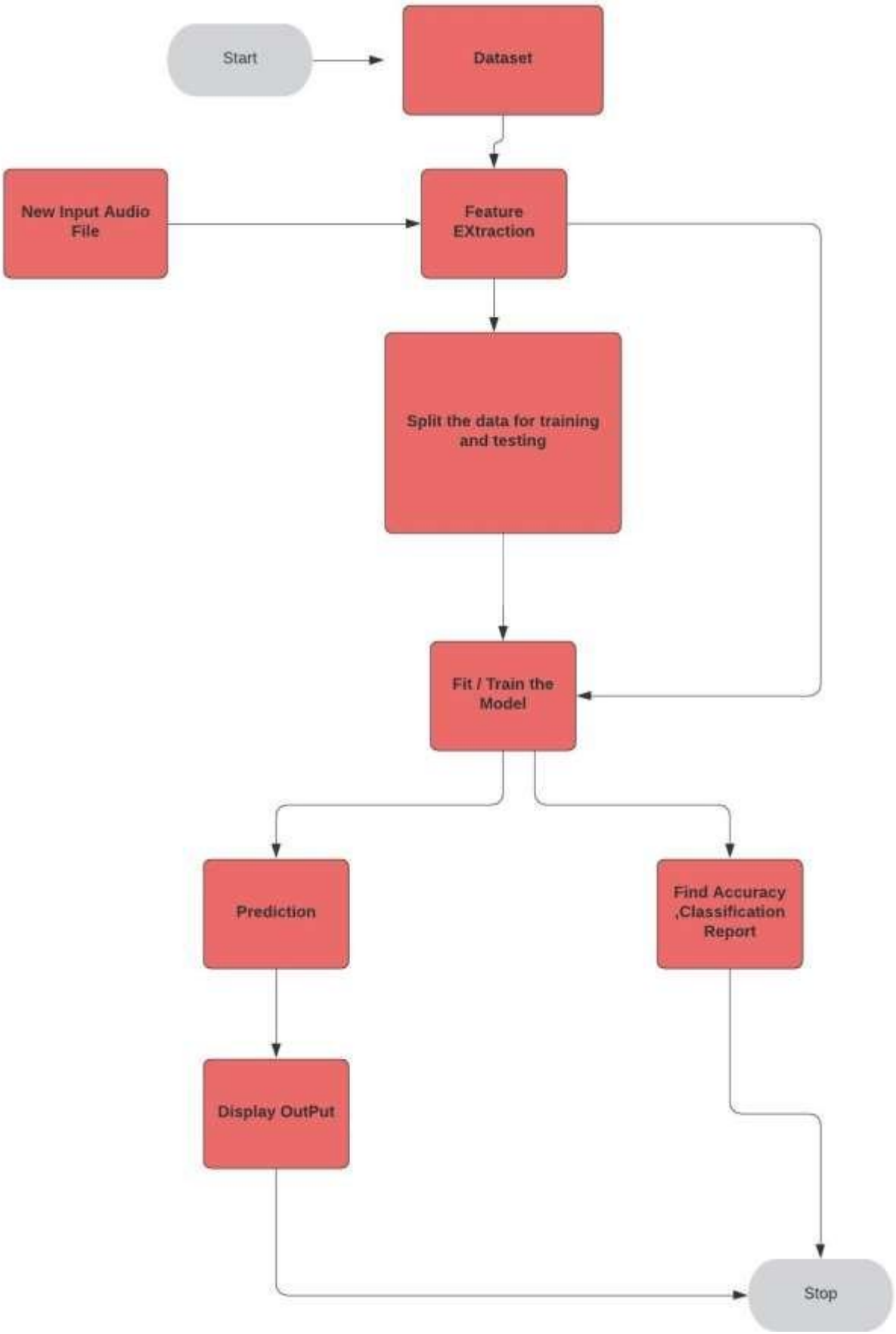
Start
Dataset
New Input Audio File
Feature EXtraction
Split the data for training and testing
Fit / Train the Model
Prediction
Find Accuracy ,Classification Report
Display OutPut
Stop

Figure 17 – System Architecture

CHAPTER 4

RESULTS AND DISCUSSION

CONFUSION MATRIX

	Predicted_angry	Predicted_sad	Predicted_neutral	Predicted_ps	Predicted_happy
True_angry	92.307693	0.000000	1.282051	2.564103	3.846154
True_sad	12.820514	67.948715	3.846154	6.410257	8.974360
True_neutral	3.846154	8.974360	82.051285	2.564103	2.564103
True_ps	2.564103	0.000000	1.282051	83.333328	12.820514
True_happy	20.512821	2.564103	2.564103	2.564103	71.794876

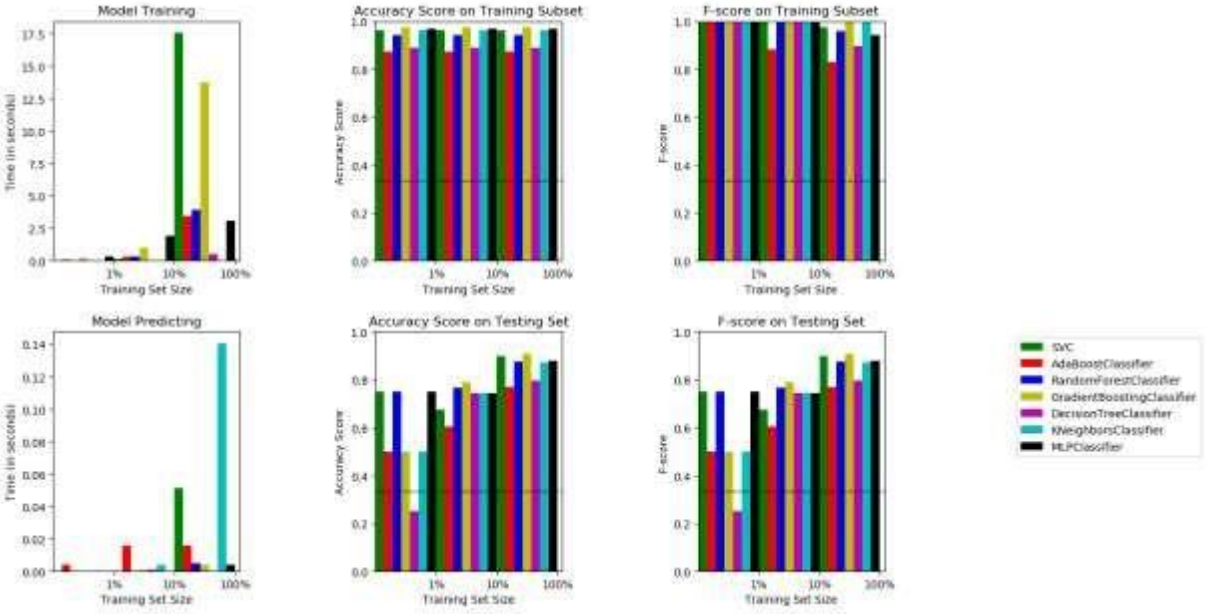

Figure 18 – Histogram on different classifiers.

CHAPTER 5

CONCLUSION AND FUTURE WORK

SER is a deeply engaging and trending topic for research in computer

science. The proposed system presents a state-of the-art algorithm for SER with real-time recognition with an accuracy. In the future, the proposed system can be extended to perform emotion recognition on multilingualism. Moreover, it can also be extended to recognize emotions at minute level with the context.

REFERENCES

[1] G. Deshmukh, A. Gaonkar, G. Golwalkar and S. Kulkarni, "Speech based Emotion Recognition using Machine Learning", *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC)*, vol. 4, no. 4, pp. 812-817, 2019.

[2] R. A. Khalil, E. Jones, M. I. Babar, T. Jan, M. H. Zafar and T. Alhussain, "Speech emotion recognition using deep learning techniques: A review", *IEEE Access*, vol. 2, no. 7, pp. 117327-117345, 2019.

[3] Y. Ü. Sonmez and A. Varol, "New Trends in Speech Emotion Recognition", *2019 7th International Symposium on Digital Forensics and Security (ISDFS)*, vol. 6, no. 8, pp. 1-7, 2019.

[4] R., Steven. Livingstone, "RAVDESS Emotional SpeechAudio", *Kaggle.Com*, 2020, [online] Available: https://www.kaggle.com/uwrfkaggler/ravdess-emotional-speech-audio.

[5] X. Chen, W. Han, H. Ruan, J. Liu, H. Li, D. Jiang, ¯Sequence-to-sequence Modelling for Categorical Speech Emotion Recognition Using Recurrent Neural Network‖, 2018 First Asian Conference on Affective Computing and Intelligent Interaction (ACII Asia), pp. 1-6, 2018.

[6] G. Liu, W. He, B. Jin, ¯Feature fusion of speech emotion recognition based on deep Learning‖, 2018 International Conference on Network Infrastructure and Digital Content (IC-NIDC), pp. 193-197, 2018.

[7] A. Rajasekhar, M. K. Hota, ¯A Study of Speech, Speaker and Emotion Recognition using Mel Frequency Cepstrum Coefficients and Support Vector Machines‖, International Conference on Communication

and Signal Processing, pp. 0114- 0118, 2018.
[8] P. Tzirakis, J. Zhang, B. W. Schuller, ¯End-to-end speech emotion recognition using deep neural networks‖, 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 5089-5093, 2018.
[9] L. Zheng, Q. Li, H. Ban, S. Liu, ¯Speech Emotion Recognition Based on Convolution Neural Network combined with Random Forest‖, The 30th Chinese Control and Decision Conference (2018 CCDC), pp. 4143-4147, 2018.

APPENDICES

SAMPLE CODE:

```
import os
def convert_audio(audio_path, target_path, remove=False):
v = os.system(f"ffmpeg -i {audio_path} -ac 1 -ar 16000 {target_path}")
    # os.system(f"ffmpeg -i {audio_path} -ac 1 {target_path}")
    if remove:
os.remove(audio_path)
    return v
def convert_audios(path, target_path, remove=False):
    for dirpath, dirnames, filenames in os.walk(path):
        for dirname in dirnames:
            dirname = os.path.join(dirpath, dirname)
            target_dir = dirname.replace(path, target_path)
            if not os.path.isdir(target_dir):
os.mkdir(target_dir)

    for dirpath, _, filenames in os.walk(path):
        for filename in filenames:
```

```
            file = os.path.join(dirpath, filename)
            if file.endswith(".wav"):
                # it is a wav file
                target_file = file.replace(path, target_path)
                convert_audio(file, target_file, remove=remove)
if __name__== "__main__":
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument("audio_path")
    parser.add_argument("target_path")
    parser.add_argument("-r", "--remove", type=bool, help="Whether to
remove the old wav file after converting", default=False)

    args = parser.parse_args()
    audio_path = args.audio_path
    target_path = args.target_path

    if os.path.isdir(audio_path):
        if not os.path.isdir(target_path):
os.makedirs(target_path)
            convert_audios(audio_path, target_path, remove=args.remove)
    elif os.path.isfile(audio_path) and audio_path.endswith(".wav"):
        if not target_path.endswith(".wav"):
            target_path += ".wav"
        convert_audio(audio_path, target_path, remove=args.remove)
    else:
        raise TypeError("The audio_path file you specified isn't appropriate
for this operation")
class DeepEmotionRecognizer(EmotionRecognizer)
```

```
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

self.n_rnn_layers = kwargs.get("n_rnn_layers", 2)
self.n_dense_layers = kwargs.get("n_dense_layers", 2)
        self.rnn_units = kwargs.get("rnn_units", 128)
self.dense_units = kwargs.get("dense_units", 128)
self.cell = kwargs.get("cell", LSTM)

        # list of dropouts of each layer
        # must be len(dropouts) = n_rnn_layers + n_dense_layers
self.dropout = kwargs.get("dropout", 0.3)
self.dropout = self.dropout if isinstance(self.dropout, list) else
[self.dropout] * ( self.n_rnn_layers + self.n_dense_layers )
        # number of classes ( emotions )
self.output_dim = len(self.emotions)

        # optimization attributes
self.optimizer = kwargs.get("optimizer", "adam")
self.loss = kwargs.get("loss", "categorical_crossentropy")

        # training attributes
self.batch_size = kwargs.get("batch_size", 64)
self.epochs = kwargs.get("epochs", 500)

        # the name of the model
        self.model_name = ""
self._update_model_name()
```

```
        # init the model
self.model = None

        # compute the input length
self._compute_input_length()

        # boolean attributes
        self.model_created = False

    def _update_model_name(self):
        """
        Generates a unique model name based on parameters passed and
put it on `self.model_name`.
        This is used when saving the model.
        """
        # get first letters of emotions, for instance:
        # ["sad", "neutral", "happy"] => 'HNS' (sorted alphabetically)
        emotions_str = get_first_letters(self.emotions)
        # 'c' for classification & 'r' for regression
        problem_type = 'c' if self.classification else 'r'
        dropout_str                =                get_dropout_str(self.dropout,
n_layers=self.n_dense_layers + self.n_rnn_layers)
        self.model_name            =            f"{emotions_str}-{problem_type}-
{self.cell.__name__}-layers-{self.n_rnn_layers}-{self.n_dense_layers}-
units-{self.rnn_units}-{self.dense_units}-dropout-{dropout_str}.h5"

    def _get_model_filename(self):
        """Returns the relative path of this model name"""
        return f"results/{self.model_name}"
```

```
    def _model_exists(self):
        """
        Checks if model already exists in disk, returns the filename,
        and returns `None` otherwise.
        """
        filename = self._get_model_filename()
        return filename if os.path.isfile(filename) else None

    def _compute_input_length(self):
        """
        Calculates the input shape to be able to construct the model.
        """
        if not self.data_loaded:
self.load_data()
self.input_length = self.X_train[0].shape[1]

    def _verify_emotions(self):
        super()._verify_emotions()
        self.int2emotions = {i: e for i, e in enumerate(self.emotions)}
self.emotions2int = {v: k for k, v in self.int2emotions.items()}

    def create_model(self):
        """
        Constructs the neural network based on parameters passed.
        """
        if self.model_created:
            # model already created, why call twice
            return
```

```
        if not self.data_loaded:
            # if data isn't loaded yet, load it
self.load_data()

        model = Sequential()

        # rnn layers
        for i in range(self.n_rnn_layers):
            if i == 0:
                # first layer
model.add(self.cell(self.rnn_units,                    return_sequences=True,
input_shape=(None, self.input_length)))
model.add(Dropout(self.dropout[i]))
            else:
                # middle layers
model.add(self.cell(self.rnn_units, return_sequences=True))
model.add(Dropout(self.dropout[i]))

        if self.n_rnn_layers == 0:
            i = 0

        # dense layers
        for j in range(self.n_dense_layers):
            # if n_rnn_layers = 0, only dense
            if self.n_rnn_layers == 0 and j == 0:
model.add(Dense(self.dense_units,                    input_shape=(None,
self.input_length)))
model.add(Dropout(self.dropout[i+j]))
```

```
            else:
model.add(Dense(self.dense_units))
model.add(Dropout(self.dropout[i+j]))

        if self.classification:
model.add(Dense(self.output_dim, activation="softmax"))
model.compile(loss=self.loss,                    metrics=["accuracy"],
optimizer=self.optimizer)
        else:
model.add(Dense(1, activation="linear"))
model.compile(loss="mean_squared_error",
metrics=["mean_absolute_error"], optimizer=self.optimizer)

        self.model = model
        self.model_created = True
        if self.verbose> 0:
print("[+] Model created")

    def load_data(self):
        super().load_data()
        # reshape X's to 3 dims
        X_train_shape = self.X_train.shape
        X_test_shape = self.X_test.shape
        self.X_train      =      self.X_train.reshape((1,      X_train_shape[0],
X_train_shape[1]))
        self.X_test       =       self.X_test.reshape((1,       X_test_shape[0],
X_test_shape[1]))

        if self.classification:
```

```
            # one-hot encode when its classification
self.y_train = to_categorical([ self.emotions2int[str(e)] for e in self.y_train
])
self.y_test = to_categorical([ self.emotions2int[str(e)] for e in self.y_test ])

        # reshape labels
        y_train_shape = self.y_train.shape
        y_test_shape = self.y_test.shape
        if self.classification:
self.y_train = self.y_train.reshape((1, y_train_shape[0],
y_train_shape[1]))
self.y_test = self.y_test.reshape((1, y_test_shape[0], y_test_shape[1]))
        else:
self.y_train = self.y_train.reshape((1, y_train_shape[0], 1))
self.y_test = self.y_test.reshape((1, y_test_shape[0], 1))

    def train(self, override=False):
        # if model isn't created yet, create it
        if not self.model_created:
self.create_model()

        # if the model already exists and trained, just load the weights and
return
        # but if override is True, then just skip loading weights
        if not override:
            model_name = self._model_exists()
            if model_name:
self.model.load_weights(model_name)
                self.model_trained = True
```

```
            if self.verbose> 0:
print("[*] Model weights loaded")
            return

        if not os.path.isdir("results"):
os.mkdir("results")

        if not os.path.isdir("logs"):
os.mkdir("logs")

        model_filename = self._get_model_filename()

self.checkpointer = ModelCheckpoint(model_filename,
save_best_only=True, verbose=1)
self.tensorboard = TensorBoard(log_dir=os.path.join("logs",
self.model_name))

self.history = self.model.fit(self.X_train, self.y_train,
                    batch_size=self.batch_size,
                    epochs=self.epochs,
                    validation_data=(self.X_test, self.y_test),
                    callbacks=[self.checkpointer, self.tensorboard],
                    verbose=self.verbose)

        self.model_trained = True
        if self.verbose> 0:
print("[+] Model trained")

    def predict(self, audio_path):
```

```
        feature = extract_feature(audio_path, **self.audio_config).reshape((1, 1, self.input_length))
        if self.classification:
            return self.int2emotions[self.model.predict_classes(feature)[0][0]]
        else:
            return self.model.predict(feature)[0][0][0]

    def predict_proba(self, audio_path):
        if self.classification:
            feature = extract_feature(audio_path, **self.audio_config).reshape((1, 1, self.input_length))
            proba = self.model.predict(feature)[0][0]
            result = {}
            for prob, emotion in zip(proba, self.emotions):
                result[emotion] = prob
            return result
        else:
            raise NotImplementedError("Probability prediction doesn't make sense for regression")



    def test_score(self):
        y_test = self.y_test[0]
        if self.classification:
            y_pred = self.model.predict_classes(self.X_test)[0]
            y_test = [np.argmax(y, out=None, axis=None) for y in y_test]
            return accuracy_score(y_true=y_test, y_pred=y_pred)
        else:
```

```
            y_pred = self.model.predict(self.X_test)[0]
            return mean_absolute_error(y_true=y_test, y_pred=y_pred)

    def train_score(self):
        y_train = self.y_train[0]
        if self.classification:
            y_pred = self.model.predict_classes(self.X_train)[0]
            y_train = [np.argmax(y, out=None, axis=None) for y in y_train]
            return accuracy_score(y_true=y_train, y_pred=y_pred)
        else:
            y_pred = self.model.predict(self.X_train)[0]
            return mean_absolute_error(y_true=y_train, y_pred=y_pred)

    def confusion_matrix(self, percentage=True, labeled=True):
        if not self.classification:
            raise NotImplementedError("Confusion matrix works only when it
is a classification problem")
        y_pred = self.model.predict_classes(self.X_test)[0]
        # invert from keras.utils.to_categorical
        y_test = np.array([ np.argmax(y, axis=None, out=None) for y in
self.y_test[0] ])
        matrix = confusion_matrix(y_test, y_pred,
labels=[self.emotions2int[e] for e in self.emotions]).astype(np.float32)
        if percentage:
            for i in range(len(matrix)):
                matrix[i] = matrix[i] / np.sum(matrix[i])
            # make it percentage
            matrix *= 100
        if labeled:
```

```
        matrix = pd.DataFrame(matrix, index=[ f"true_{e}" for e in
self.emotions ],
                              columns=[ f"predicted_{e}" for e in self.emotions ])
        return matrix

    def get_n_samples(self, emotion, partition):
        if partition == "test":
            if self.classification:
                y_test = np.array([ np.argmax(y, axis=None, out=None)+1 for
y in np.squeeze(self.y_test) ])
            else:
                y_test = np.squeeze(self.y_test)
            return len([y for y in y_test if y == emotion])
        elif partition == "train":
            if self.classification:
                y_train = np.array([ np.argmax(y, axis=None, out=None)+1 for
y in np.squeeze(self.y_train) ])
            else:
                y_train = np.squeeze(self.y_train)
            return len([y for y in y_train if y == emotion])

    def get_samples_by_class(self):
        train_samples = []
        test_samples = []
        total = []
        for emotion in self.emotions:
            n_train      =      self.get_n_samples(self.emotions2int[emotion]+1,
"train")
            n_test = self.get_n_samples(self.emotions2int[emotion]+1, "test")
```

```
            train_samples.append(n_train)
            test_samples.append(n_test)
total.append(n_train + n_test)

        # get total
total.append(sum(train_samples) + sum(test_samples))
        train_samples.append(sum(train_samples))
        test_samples.append(sum(test_samples))
        return pd.DataFrame(data={"train": train_samples, "test":
test_samples, "total": total}, index=self.emotions + ["total"])

    def get_random_emotion(self, emotion, partition="train"):
        """
        Returns random `emotion` data sample index on `partition`
        """
        if partition == "train":
            y_train = self.y_train[0]
            index = random.choice(list(range(len(y_train))))
            element = self.int2emotions[np.argmax(y_train[index])]
            while element != emotion:
                index = random.choice(list(range(len(y_train))))
                element = self.int2emotions[np.argmax(y_train[index])]
        elif partition == "test":
            y_test = self.y_test[0]
            index = random.choice(list(range(len(y_test))))
            element = self.int2emotions[np.argmax(y_test[index])]
            while element != emotion:
                index = random.choice(list(range(len(y_test))))
                element = self.int2emotions[np.argmax(y_test[index])]
```

```
        else:
            raise TypeError("Unknown partition, only 'train' or 'test' is accepted")

        return index

    def determine_best_model(self):
        # TODO
        # raise TypeError("This method isn't supported yet for deep nn")
        pass


if __name__== "__main__":
    rec = DeepEmotionRecognizer(emotions=['angry', 'sad', 'neutral', 'happy', 'disgust', 'boredom','fear'],
                                epochs=300, verbose=0)
rec.train(override=False)
print("Test accuracy score:", rec.test_score() * 100, "%")

def get_label(audio_config):
    features = ["mfcc", "chroma", "mel", "contrast", "tonnetz"]
    label = ""
    for feature in features:
        if audio_config[feature]:
            label += f"{feature}-"
    return label.rstrip("-")


def get_dropout_str(dropout, n_layers=3):
```

```
    if isinstance(dropout, list):
        return "_".join([ str(d) for d in dropout])
    elif isinstance(dropout, float):
        return "_".join([ str(dropout) for i in range(n_layers) ])


def get_first_letters(emotions):
    return "".join(sorted([ e[0].upper() for e in emotions ]))


    mfcc = kwargs.get("mfcc")
    chroma = kwargs.get("chroma")
    mel = kwargs.get("mel")
    contrast = kwargs.get("contrast")
    tonnetz = kwargs.get("tonnetz")
    try:
        with soundfile.SoundFile(file_name) as sound_file:
            pass
    except RuntimeError:
        # not properly formated, convert to 16000 sample rate & mono
channel using ffmpeg
        # get the basename
        basename = os.path.basename(file_name)
dirname  = os.path.dirname(file_name)
        name, ext = os.path.splitext(basename)
        new_basename = f"{name}_c.wav"
        new_filename = os.path.join(dirname, new_basename)
        v = convert_audio(file_name, new_filename)
        if v:
```

```
            raise NotImplementedError("Converting the audio files failed, make sure `ffmpeg` is installed in your machine and added to PATH.")
        else:
            new_filename = file_name
        with soundfile.SoundFile(new_filename) 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


def get_best_estimators(classification):
    if classification:
        return pickle.load(open("grid/best_classifiers.pickle", "rb"))
    else:
        return pickle.load(open("grid/best_regressors.pickle", "rb"))


def get_audio_config(features_list):
    audio_config = {'mfcc': False, 'chroma': False, 'mel': False, 'contrast':
False, 'tonnetz': False}
    for feature in features_list:
        if feature not in audio_config:
            raise TypeError(f"Feature passed: {feature} is not recognized.")
        audio_config[feature] = True
    return audio_config
from emotion_recognition import EmotionRecognizer
import pydub
import pyaudio
import os
import wave
from sys import byteorder
from array import array
from struct import pack
from sklearn.ensemble import GradientBoostingClassifier,
```

BaggingClassifier

```
from utils import get_best_estimators
import streamlit as st
import pickle
import numpy as np
from utils import get_best_estimators

THRESHOLD = 500
CHUNK_SIZE = 1024
FORMAT = pyaudio.paInt16
RATE = 16000

SILENCE = 30



def get_estimators_name(estimators):
    result = [ '"{}"'.format(estimator.__class__.__name__) for estimator, _,
_ in estimators ]
    return ','.join(result), {estimator_name.strip(""): estimator for
estimator_name, (estimator, _, _) in zip(result, estimators)}

if __name__== "__main__":
    estimators = get_best_estimators(True)
    estimators_str, estimator_dict = get_estimators_name(estimators)
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument("-e", "--emotions", default="sad,neutral,happy")
    parser.add_argument("-m", "--model", default="BaggingClassifier")
```

```
st.title('SPEECH EMOTION DETECTION')
    # Parse the arguments passed
    args = parser.parse_args()

    features = ["mfcc", "chroma", "mel"]
    detector = EmotionRecognizer(estimator_dict[args.model],
emotions=args.emotions.split(","), features=features, verbose=0)
detector.train()
    #print("Test accuracy score:
{:.3f}%".format(detector.test_score()*100))
    #print("Emotion of Mentioned wav file is : ")
    filename = st.file_uploader("Choose a file")
    if filename is not None:
        if filename.name.endswith('wav'):
            audio = pydub.AudioSegment.from_wav(filename)
            file_type = 'wav'
        elif filename.name.endswith('mp3'):
            audio = pydub.AudioSegment.from_mp3(filename)
            file_type = 'mp3'

audio.export("test.wav", format=file_type)
    #st.write(filename[1])
    filename = "test.wav"
    #record_to_file(filename)
    if st.button("predict"):
        result = detector.predict(filename)
st.success(result)
```

SCREEN SHOTS:

Figure 19 -Cammand Prompt Output Screenshot

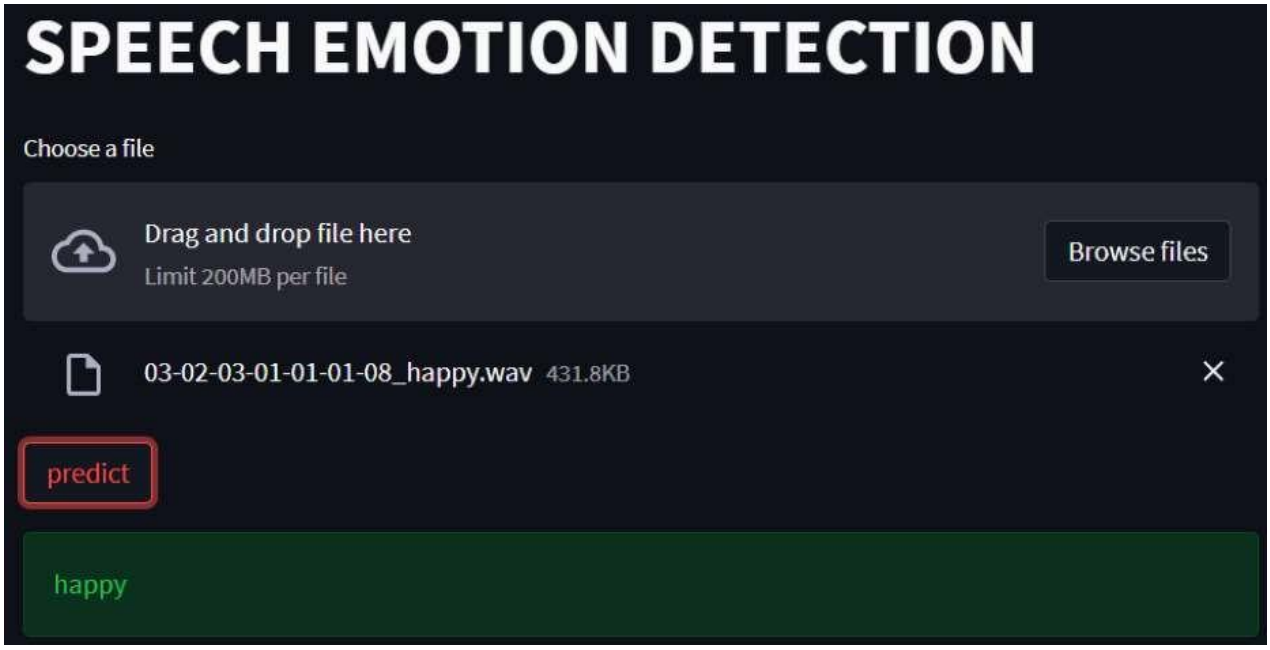

Figure 20 - Output Screenshot of Happy wav file

Figure21 - Output Screenshot of Sad wav file

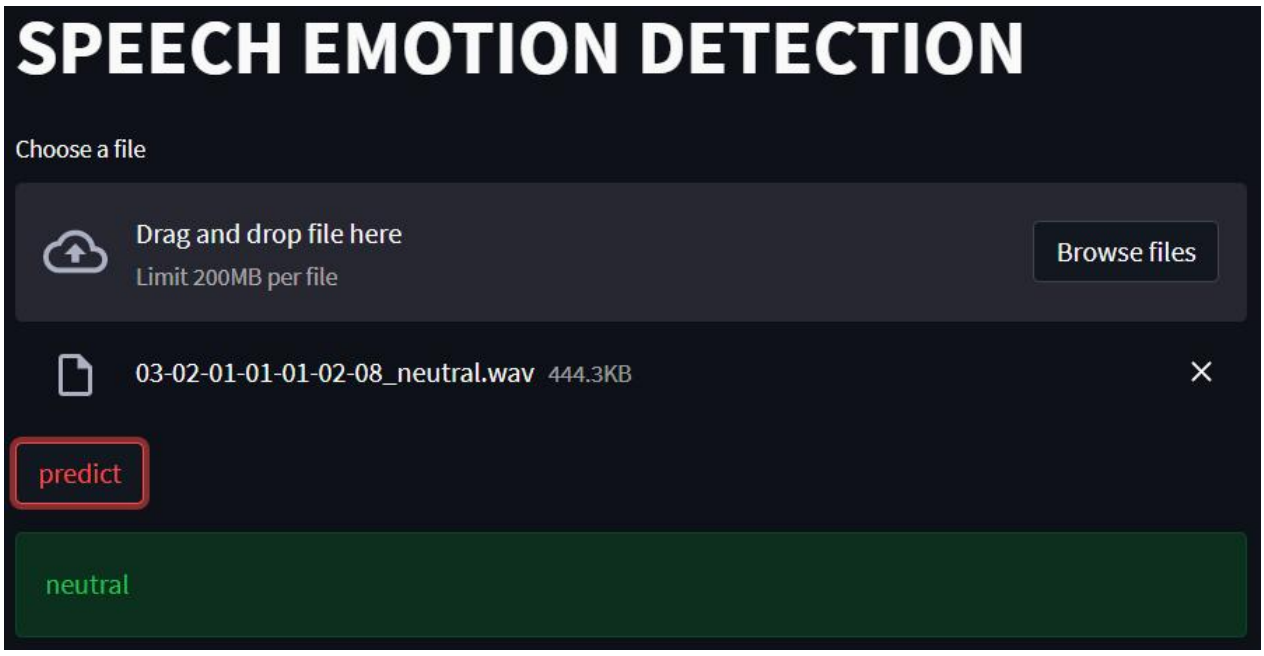

Figure 22 - Output Screenshot of Neutral wav file