**AN OPTIMIZED FUZZY BASED PERSONALIZED DEEP**

**GLUCOSE LEVEL PREDICTION FOR TYPE 1 DIABETES**

**PATIENTS AND SEVERITY ESTIMATION**

**A PROJECT REPORT**

*Submitted by*

|  |  |
| --- | --- |
| **MUTHUKUMARASAMY S** | **(Reg. No. 202104088)** |
| **NAREN KARTHIKEYAN M** | **(Reg. No. 202104092)** |

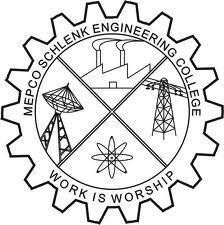
*in partial fulfillment for the award of the degree*

*of*

**BACHELOR OF ENGINEERING**

## in

**COMPUTER SCIENCE AND ENGINEERING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI**

## (An Autonomous Institution affiliated to Anna University Chennai)

**April 2025**

## BONAFIDE CERTIFICATE

Certified that this project report titled **AN OPTIMIZED FUZZY BASED PERSONALIZED DEEP GLUCOSE LEVEL PREDICTION FOR TYPE 1 DIABETES PATIENTS AND SEVERITY ESTIMATION** is the bonafide work of **Mr. S.MUTHUKUMARASAMY (Reg. No.: 202104088), Mr. M.NAREN KARTHIKEYAN (Reg. No.: 202104092)**

who

carried out the research under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

## Internal Guide Head of the Department

**Dr. S.Amutha, M.E., Ph.D**. **Dr. J. RAJA SEKAR, M.E., Ph.D**.

Associate Professor, Professor and Head,

Dept. of Computer Science & Engg, Dept. of Computer Science & Engg, Mepco Schlenk Engineering College, Mepco Schlenk Engineering College, (Autonomous) (Autonomous)

Sivakasi – 626 005. Sivakasi – 626 005.

Submitted for Viva-Voce Examination held at **Mepco Schlenk Engineering College (Autonomous), Sivakasi** on / / 2025 **.**

**INTERNAL EXAMINER EXTERNAL EXAMINER**

## ABSTRACT

The accurate and personalized blood glucose level predictions play the key role in Type 1 diabetes to avoid serious complications. The accuracy is very low, but traditional physiological models better explain the approach; however, they are not flexible. Deep learning models are good at picking up the patterns that are present in the historical data but not good at interpreting the data. In this work, Hybrid method is proposed that took advantage of both the models in such a way that they provide accurate predictions. Physiological white box model inspired by the UVA/Padova T1D simulator is combined with black box methods as LSTM, GRU and TCN, and methodology is developed to integrate them. Thus, this data relies on continuous glucose monitoring (CGM), intake of insulin, and consumption of carbohydrates processed to some extent such as moving averages, rate of change of glucose, and lags, etc. To address this, physiological model subsystems, like Insulin absorption, glucose absorption, and glucose-insulin kinetics, are integrated with the integration using deep learning methods, Long Short Term Memory (LSTM), Gated Recurrent Units (GRU), and Temporal Convolution Networks (TCN). Using physiological models to provide glucose absorbed to bloodstream improves the predictability of the black box model. Additionally, the genetic algorithm is applied for the optimization of the prediction time horizon and HbA1c method for estimating the average blood glucose level. However, in this case, a fuzzy algorithm is also used which predicts the severity level based on glucose levels and weight as inputs involving 20 rules with some combinations to enhance the prediction accuracy. Such data can also be used for real-world data for enhancing patient outcomes.

## 

## ACKNOWLEDGEMENT

First and foremost, we thank the **LORD ALMIGHTY** for his abundant blessings that are shown upon our past, present, and future successful endeavours.

We extend our sincere gratitude to our college management and Principal **Dr.S.Arivazhagan M.E., Ph.D.,** for providing a sufficient working environment such as systems and library facilities. We also thank him very much for providing us with adequate lab facilities, which enable us to complete the project.

We would like to extend our heartfelt gratitude to   
**Dr.J.Raja Sekar M.E., Ph.D., Professor and Head**, Department of Computer Science and Engineering, Mepco Schlenk Engineering College for giving us the golden opportunity to undertake a project of this nature and for this most valuable guidance given at every phase of our work.

We would also like to extent our gratitude to   
**Dr.P.Vinoth M.E., Ph.D., Assistant Professor (Sl.G.)**, Department of Computer Science and Engineering, for being our Project Coordinator and directing us throughout our project.

We would also like to extend our gratitude to   
**Dr.S.Amutha M.E., Ph.D., Associate Professor,** Department of Computer Science and Engineering, Mepco Schlenk Engineering College for being our Project Guide and directing us throughout the project. She has put her valuable experience and expertise into supporting and suggesting us throughout the Project to bring out the best.

Our sincere thanks to our revered **faculty members and lab technicians**

for their help over this project work.

Last but not least, we extend our indebtedness to our beloved **family and our friends** for their moral support which made the project a successful one.

v

**TABLE OF CONTENTS**

**CHAPTE****R NO.**

**CONTENTS PAGE**

**NO.**

[**ABSTRACT** iii](#_TOC_250011)

[**ACKNOWLEDGEMENT** iv](#_TOC_250010)

**LIST OF TABLES** ix

**LIST OF FIGURES** x

[**LIST OF ABBREVIATIONS** x](#_TOC_250009)i

1. **INTRODUCTION 1**
   1. [Problem Description 1](#_TOC_250008)
   2. Objectives of the Project 1
   3. [Outcomes of the Project 2](#_TOC_250007)
   4. [Scope of the Project 2](#_TOC_250006)
   5. Report review 3
2. **LITERATURE SURVEY 4**
   1. [Overview 4](#_TOC_250005)
   2. Individualized Models for Glucose Prediction in Type 1 4

Diabetes: Comparing Black-Box Approaches to a

Physiological White-Box One

* 1. The Ohio T1DM Dataset For Blood Glucose Level 5

Prediction Update 2020

* 1. Deep Physiological Model for Blood Glucose Prediction 6

In T1DM Patients

* 1. Blood Glucose Prediction Model For Typer 1 Diabetes Based 7

On Artificial Neural Network with Time-Domain Features

* 1. A Comparison of Machine Learning Algorithms For Diabetes 7

Prediction

* 1. A Personalized Blood Glucose Level Prediction Model With a 8
  2. Blood Glucose Prediction With Deep Neural Networks 9

Using Weighted Decision Level Fusion

* 1. An Ensemble Machine Learning Approach For Predicting 10

Type -1 Diabetes Mellitus Based on Lifestyle Indicators.

vi

|  |  |  |
| --- | --- | --- |
| **CHAPTER** | **CONTENTS** | **PAGE** |
| **NO.** |  | **NO.** |
| 2.10 | Short-Term Prediction Method Of Blood Glucose Based On Temporal Multi-Head Attention Mechanism For Diabetic Patients | 11 |
| 2.11 | A Novel Machine Learning Approach For Diagnosing Diabetes With A Self-Explainable Interface | 12 |
| 2.12 | Enhanced Blood Glucose Levels Prediction With A Smartwatch | 13 |
| 2.13 | Generative Adversarial Networks For Improved Glucose Forecasting | 14 |
| 2.14 | Meta-Learning Methods For Limited Data Glucose Prediction | 15 |
| 2.15 | Domain-Specific Constraints In Generalized Multivariate Glucose Forecasting | 15 |
| 2.16 | LSTM Networks For Predicting Blood Glucose In Time Series Data | 16 |
| 2.17 | LSTM And Attention-Based Models For Blood Glucose Prediction Comparatively Analyzed | 17 |
| 2.18 | Type 1 Diabetes Patients' Computational Identification Tool | 18 |
| 2.19 | Bayesian Parameter Estimation for Type 1 Diabetes Modeling | 18 |
| 2.20 | Multi-Subject Learning For Enhanced Blood Glucose Prediction | 19 |
| 2.21 | Mealtime Insulin Bolus Prediction Ensemble Method | 20 |
| 2.22 | Summary | 21 |
|  |  |  |

## SYSTEM STUDY 22

* 1. [Overview 22](#_TOC_250004)
  2. Existing System 22
  3. [Steps Involved In Glucose Level Prediction in Existing System 23](#_TOC_250003)
  4. Proposed System 23
  5. Steps Involved in Proposed System 25
  6. Procedure for Physiological White Box Models 25

|  |  |  |
| --- | --- | --- |
| **CHAPTER** | **CONTENTS** | **PAGE** |
| **NO.** |  | **NO.** |

* 1. Procedure for Deep Learning(Black Box) Models 25
  2. Procedure for Building An Hybrid Model 26
  3. Procedure for HbA1c And Fuzzy Based Severity Order 26

Calculation

* 1. Summary 26

## SYSTEM DESIGN 27

* 1. [Overview 27](#_TOC_250004)
  2. System Architectural Design 28
  3. Module Description 29
  4. [Summary 31](#_TOC_250003)

1. **SYSTEM IMPLEMENTATION 32**
   1. [Data Collection 32](#_TOC_250001)
   2. Dataset Description 33
   3. Data Preprocessing 34
   4. Feature Extraction 35
   5. Traditional Physiological Model 35
   6. Black Box Deep Learning Model 39
   7. Hybrid Integration 42
   8. Genetic PH Optimization 42
   9. Hba1C Calculation 43
   10. Fuzzy Order Severity Classification 44
   11. Summary 46
2. **RESULTS AND DISCUSSION 47**
   1. [Estimating The Best Parameter From Adaptive SCMH](#_TOC_250001) 47

6.2 Hybrid Model Integration Prediction 48

|  |  |  |  |
| --- | --- | --- | --- |
| **CHAPTER**  **NO** |  | **CONTENTS** | **PAGE**  **NO.** |

6.3 [HbA1c Prediction](#_TOC_250001) 50

* 1. Fuzzy Rule Based Severity Estimation 50
  2. Hybrid Model for Predicting Future Glucose Level 51
  3. State Of Art Of Comparison 56
  4. [Summary 58](#_TOC_250001)

7 CONCLUSION AND FUTURE WORK 59

7.1 [Conclusion](#_bookmark13) 59

7.2 [Future Work](#_bookmark13) 59

7.3 [Social Impacts](#_bookmark13) 60

7.4 [Applicability of the Project](#_bookmark13) 61

APPENDIX I 62

APPENDIX 2 63

[REFERENCES](#_bookmark15) 82

|  |  |  |
| --- | --- | --- |
|  | **LIST OF TABLES** |  |
| **TABLE NO.** | **TABLE CAPTION** | **PAGE NO.** |
| 6.1 | Sample Best Parameters From Adaptive SCMH-1 | 47 |
| 6.2 | Sample Best Parameters From Adaptive SCMH-2 | 48 |
| 6.3 | Sample LSTM Prediction | 48 |
| 6.4 | Sample GRU Prediction | 49 |
| 6.5 | Sample TCN Prediction | 49 |
| 6.6 | Sample HbA1c Prediction | 50 |
| 6.7 | Sample Fuzzy Severity Order Prediction | 50 |
| 6.8 | Sample Performance Comparison for LSTM – 12,34,43 Minute Time Horizon | 51 |
| 6.9 | Sample Performance Comparison for GRU – 12,34,43 Minute Time Horizon | 52 |
| 6.10 | Sample Performance Comparison for TCN – 12,34,43 Minute Time Horizon | 52 |
| 6.11 | A personalized blood glucose level prediction model with a fine-tuning strategy | 56 |
| 6.12 | Blood glucose prediction with deep neural networks using weighted decision level fusion | 56 |
| 6.13 | Continuous Glucose, Insulin and Lifestyle Data Augmentation in Artificial Pancreas Using *Adaptive* Generative and Discriminative Models | 57 |
| 6.14 | Individualized Models for Glucose Prediction in Type 1 Diabetes: Comparing Black-Box Approaches to a Physiological White-Box One | 57 |
| 6.15 | An Optimized Fuzzy based Personalized DeepGlucose Level Prediction and Severity | 58 |
|  |  |  |
|  | **LIST OF FIGURES** |  |
| **FIGURE NO.** | **FIGURE CAPTION** | **PAGE NO.** |
| 4.1 | Proposed System Design | 27 |
| 5.1 | Continuous Glucose Monitor | 32 |
| 5.2 | Insulin | 33 |
| 5.3 | LSTM Architecture | 40 |
| 5.4 | GRU Architecture | 41 |
| 5.5 | TCN Architecture | 42 |
| 6.1 | Sample Performance Comparison for LSTM, GRU, TCN for 34 minute horizon | 53 |
| 6.2 | Sample Performance Comparison for LSTM, GRU, TCN for 34 minute horizon | 53 |
| 6.3 | Sample Performance Comparison for LSTM, GRU, TCN for 34 minute horizon | 54 |
| 6.4 | RMSE Comparison of Proposed and Existing Approach | 55 |
|  |  |  |

## LIST OF ABBREVIATIONS

**ABBREVIATIONS EXPANSION**

**SCMH S**ingle **C**hain **M**etropolis **H**astings

**LSTM L**ong **S**hort **T**erm **M**emory

**GRU G**ated **R**ecurrent **U**nit

**TCN T**emporal **C**onvolutional **N**eural **N**etwork

**T1DM T**ype-1 **D**iabetes **M**ellitus

**CGM C**ontinuous **G**lucose **M**onitoring

**RMSE R**oot **M**ean **S**quare **E**rror

**MAE M**ean **A**bsolute **E**rror

**MSE M**ean **S**quare **E**rror

## CHAPTER 1 INTRODUCTION

## PROBLEM DESCRIPTION

The frequency of diabetes continues to increase because at least one person in every family can develop this condition. The condition develops when the body either fails to use insulin well or makes insufficient insulin levels. Insulin as a pancreatic hormone helps regulate blood sugar because it absorbs glucose to provide energy. High blood sugar occurs when insulin does not work properly creating severe medical complications.

The three distinct diabetes conditions are gestational diabetes and Type 1 and Type 2 diabetes. The immune system destroys insulin-producing cells during Type 1 diabetes which creates the need for permanent insulin injections. Gestational diabetes arises from pregnancy and puts women at risk of developing future Type 2 diabetes although Type 2 most commonly occurs when someone is overweight or inactive.

The main objective of this study involves determining blood glucose levels among patients who have Type 1 diabetes. The dark patterns of black-box models remain uninterpretable yet discover hidden patterns while traditional physiological models provide insightful yet rigid and imprecise data predictions. The predictive accuracy gets improved when black-box models including LSTM and GRU with TCN are combined with Adaptive Single Component Metropolis Hastings implementation of physiological white-box models. The extraction of important elements takes place through simulated data together with genuine patient measurements obtained from CGM products.

The assessment of HbA1c enables the determination of averaged glucose levels throughout three months which supports early intervention decisions. The fuzzy logic algorithm employs glucose and weight parameters as input when processing 20 predefined rules to generate reliable diabetes management. Patient results improve because the combination method provides both precise interpretations and accurate readings.

## OBJECTIVES OF THE PROJECT

This study aims to develop a deep learning-based predictive model for glucose level forecasting in order to enhance diabetes care with accurate and timely forecasts. The plan includes an algorithm that helps patients and healthcare providers make decisions. Tools for data visualization will highlight trends and patterns to improve understanding of glucose variations. The reliability of the model will be tested using real-world datasets. It also uses Genetic Algorithm(GA) for optimizing the time horizon which increases the accuracy. Moreover, fuzzy logic-based severity detection will classify the degree of diabetes, and HbA1c calculations will predict future average glucose trends, enabling customized healthcare solutions. The system combines machine learning and physiological models to ensure interpretability and accuracy. Ongoing observation and forecasting will help with early intervention and better treatment planning. Improving patient outcomes and reducing diabetes-related complications are the ultimate goals of this research.

## OUTCOMES OF THE PROJECT

Three main goals are the focus of this study. Its primary goal is to create a deep learning model for blood glucose prediction that is both highly accurate and flexible, and that has been verified using actual patient data. Second, to ensure accurate and timely glucose predictions, an algorithm will be created to gather and preprocess CGM data, insulin dosages, and food intake. In order to improve forecasting accuracy, a Genetic Algorithm (GA) will also be used to optimize the time horizon. In order to improve individualized diabetes care, a fuzzy-based severity detection system will categorize each patient's glucose level severity and a HbA1c calculation will be used to forecast future average glucose trends.

## OUTLINE OF THE PROJECT

This work explores machine learning-based glucose prediction for Type 1 Diabetes (T1D), with a focus on integrating deep learning models (black-box) and physiological models (white-box). The main goal is to use the complementing benefits of Long Short-Term Memory (LSTM), Gated Recurrent Units (GRU), and Temporal Convolutional Networks (TCN) by combining physiological subsystems (glucose absorption, insulin absorption, and glucose-insulin kinetics). A genetic algorithm is utilized to dynamically modify the prediction horizon in order to increase forecast accuracy, while fuzzy logic classifies disease severity based on patient weight and glucose levels. The hybrid approach aims to generate more accurate and understandable glucose estimates than single techniques. Measures such as Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and The R2 score of the OhioT1DM dataset is based on real-time CGM, insulin, and meal data. Additionally, by predicting long-term glucose changes, HbA1c calculations enhance clinical relevance.

## ORGANIZATION OF THE PROJECT

The goal of this study is to create a hybrid machine learning approach for accurate blood glucose prediction in individuals with Type 1 Diabetes. The project begins with an overview of Type 1 Diabetes and a review of relevant studies on glucose prediction methods. The researchers build a novel hybrid model that blends deep learning methods (LSTM, GRU, and TCN) with physiological white-box models after preprocessing the time-series data and collecting CGM, insulin, and meal data from the OhioT1DM dataset.

The study employs a genetic algorithm to dynamically determine prediction boundaries for optimal glucose forecasting and fuzzy logic for severity classification. The paper compares the hybrid strategy with standalone methods and evaluates model performance using RMSE, MAE, and R2 measures. In order to improve patient outcomes, the conclusion highlights the clinical significance of precise glucose estimates and offers suggestions for future research initiatives, including real-world application and integration with diabetes care systems.

## CHAPTER 2 LITERATURE SURVEY

## OVERVIEW

## This literature review explores how blood glucose level can be predicted for Type 1 diabetic patients with a preference to a hybrid method incorporating physiological white box models with the deep learning methods. Then it describes methods that were used in earlier studies of prediction of blood sugar levels from Continuous Glucose Monitoring (CGM), insulin intake, carbohydrate intake. To the best of my knowledge, the purpose of the review is to determine how well different machine learning algorithms, Temporal Convolution Networks (TCN), Gated Recurrent Units (GRU), and Long Short-Term Memory (LSTM) predict blood glucose level. Also, it discuss how fuzzy logic systems can be utilized to classify the degree of diabetes dependent on patient weight and glucose levels, how genetic algorithms may be combined to enhance the prediction time horizon. The literature review also covers on performance metrics used to continuous analyze the precision and dependability of the glucose prediction model, including Mean Relative Absolute Error (MAE), Root Mean Squared Error (RMSE), and coefficient of determination (R2 Score). Considering all, the survey, helps in development of personalized and trustworthy prediction systems that help boosting the diabetes care and as the end result improves the patient outcomes and the quality of living of people with T1D.

## INDIVIDUALIZED MODELS FOR GLUCOSE PREDICTION IN TYPE 1 DIABETES: COMPARING BLACK-BOX APPROACHES TO A PHYSIOLOGICAL WHITE-BOX ONE

**AUTHORS NAME**

Giacomo Cappon *et.al.*

## METHODOLOGY

This work combines adaptive strategies with Black-box and White-box models in order to predict glucose levels in diabetes management. Black box approaches use machine learning techniques which model complex relationship in data without any physiological knowledge needed; these approaches are used in order to capture nonlinear patterns from large datasets. In contrast, "white-box" structured physiological models that model biological processes assist to understand the causes of the phenomena of glucose regulation. The rARX model is an adaptive technique that increases the prediction accuracy and adapt to the changing real time conditions necessary for real time diabetes management through the dynamic updating of parameters based on real time CGM data.

## MERITS

* Both the White-box and the Black-box allow some layer of insights into how glucose or patients respond, with White-box being more valuable for understanding and the Blackbox offering high prediction accuracy and learning efficiency from larger datasets.
* To increase model’s real-time responsiveness, adaptive strategies become better as a patient’s condition changes, making room for dynamic adjustments to better manage.
* Together, these represent a better, more accurate method of predicting the body, and an improved comprehension of the body.

## DEMERITS

## Black box models are not interpretable, and this makes clinicians uncomfortable about their predictions.

## Although their insightfulness, by nature, white box models require thorough parameter tuning, which might take a significant amount of time and resources.

## In addition, the quality of CGM data is important for adaptive strategies, since faulty data entry may produce predictions that are less accurate, and thereby endanger real-time diabetes management.

## THE OHIO T1DM DATASET FOR BLOOD GLUCOSE LEVEL PREDICTION UPDATE 2020

**AUTHORS NAME**

Cindy Marling1 & Razvan Bunescu1

## METHODOLOGY

In eight weeks, 12 people with Type 1 diabetes had their insulin and continuous glucose monitoring (CGM) data collected. Physiological monitoring with fitness bands such activity and heart rate was recorded with participants wearing them. Moreover, recording self reported data about life events which have affected glucose levels in the smart phone app allow insights of participants daily routines and events, which may influence glucose regulation.

## MERITS

* The combined data set allows the use of a more comprehensive approach as it includes multiple data types, ICE, insulin data, physiological metrics, and self reported life events.
* Longitudinal 8 week duration allows for observation of trends and patterns in glucose levels in time and this provides the opportunity to observe potential predictors of glucose fluctuations and management of diabetes.

## DEMERITS

* As the sample size consists of only 12 participants, the findings might not be generalized to the entire population.
* The accuracy of the predictions and insights that result from the study may also be affected by lack of or skewed analysis due to data gaps, particularly the missing overnight data from some participants.

**2.4 DEEP PHYSIOLOGICAL MODEL FOR BLOOD GLUCOSE PREDICTION IN T1DM PATIENTS**

**AUTHORS NAME**

Mario Munoz-Organero

## METHODOLOGY

## This study develops a deep learning model in order to predict blood glucose levels in patients with Type 1 diabetes (T1DM) utilizing physiological data. By using deep neural networks to model the level of intricacy that physiological interactions on glucose levels, a more thorough and precise prediction framework is created. The ability of the model to accurately predict variations in glucose level for each physiological factor is improved, and it can capture nonlinear relationships between multiple physiological factors.

## MERITS

* Using a deep learning method, the glucose predictions are extremely accurate as it can successfully capture intricate physiological interactions.
* It provides deeper understanding on nonpathophysiological regulatory factors of blood glucose level in T1DM patients.
* Additionally, complex modeling techniques enable the improvement of accuracy in prediction in turn enhancing diabetes management and tailored treatment plans.

## DEMERITS

## In order for the model to be easily trained however, we require a relatively large amount of high quality physiological data and hence data availability is important for performance.

## Deep learning models are complicated and require computation and optimization work in order to run, hence they require large amounts of processing power.

## Furthermore, since different patient populations with dissimilar physiological responses can also result in difficulty generalizing the model to the physical world, it may not be used in actual clinical settings.

## 2.5 BLOOD GLUCOSE PREDICTION MODEL FOR TYPE 1 DIABETES BASED ON ARTIFICIAL NEURAL NETWORK WITH TIME-DOMAIN FEATURES

**AUTHORS NAME**

Ganjar Alfian *et.al.*

## METHODOLOGY

Time domain features are used in this study to predict the blood glucose levels through Artificial Neural Network (ANN) model. Glucose trends and patterns over time are captured by the model in order to achieve more accurate and reliable predictions. It appears that the ANN framework can successfully learn from historical glucose values in order to predict upcoming glucose fluctuations using sequential glucose data, which in turn will help in diabetes management.

## MERITS

* Prediction Accuracy of the ANN model is enhanced by paying attention to time domain features that provide better identification of glucose trends.
* It is effective in patterns storing long time axes, which makes it perfect for continuous glucose monitoring activities.
* The ANN model is robust in its performance, resulting in the ability to generate reliable predictions that may be leveraged in making the decisions regarding diabetes management.

## DEMERITS

## Data availability is a critical challenge because the model needs continuous, high quality glucose data in order to be trained effectively.

## ANN models are computationally intensive and complex, consequently it demands a lot of computation otherwise their trained models and symbols would require a lot of computational operation.

## The model may not capture all variability in a person’s response to glucose due to which in personalized diabetes management, it may not be as accurate.

## A COMPARISON OF MACHINE LEARNING ALGORITHMS FOR DIABETES PREDICTION

**AUTHORS NAME**

Jobeda Jamal Khanam & Simon Y. Foo

## METHODOLOGY

This study evaluates a number of machine learning algorithms in order to determine the best way to predict diabetes. Each algorithm’s performance is assessed with standard metrics like accuracy, precision, recall to get a comprehensive analysis of how any given model predicts diabetes outcomes. This comparison serves to highlight the advantages and disadvantages of each algorithm so that diabetes prediction decision making is improved.

## MERITS

* Based on some performance metrics like accuracy, precision and recall, the study makes a complete comparison of different machine learning algorithms.
* It helps to identify the most successful one.
* It provides some of the information on what works for diabetes prediction and what doesn’t so that we can chose the right algorithm (i.e: will it help us with diabetes prediction or not).

## DEMERITS

* The algorithms’ performances also are affected by the quality and features of the dataset used, and hence, can contribute to the outcomes.
* The results could be limited because the study did not take in account the specific tuning and optimization procedures required for each algorithm.
* Computing many algorithms is a significant amount of computing power and may also be a deterrent when implemented in real time or in large scale.

## A PERSONALIZED BLOOD GLUCOSE LEVEL PREDICTION MODEL WITH A FINE-TUNING STRATEGY: A PROOF-OF-CONCEPT STUDY

**AUTHORS NAME**

Wonju Seoa *et al.*

## METHODOLOGY

Among these, the study is also able to gather real world continuous glucose monitoring (CGM) data along with contextual data like when the patient consumed their food, when they were active, and when they administered insulin. A first model is trained with regression models and neural networks on a generic dataset. A custom loss function that takes into account variability in glucose response is added and transfer learning is employed to improve personalization by refining the model using patient data. The model is evaluated in term of metrics such as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE), and baselines are compared.

## MERITS

* Prediction accuracy is increased significantly while also making the model more suitable for each particular data of each patient.
* In fact, the contextual features such as dose of insulin, physical activity, or intake of meal improve the accuracy and relevancy of glucose level predictions as compared to other models.
* In the process of strengthening the flexibility of model’s response to individual variability this method improves real time diabetes management.

**DEMERITS**

* It is hard to maintain the model's extensive and ongoing patient data collection over long periods of time.
* The use of smart machine learning techniques requires investing a lot of time and knowledge.
* This model may not always perform well if there are insufficient patient specific data available.

**2.8 BLOOD GLUCOSE PREDICTION WITH DEEP NEURAL NETWORKS USING WEIGHTED DECISION LEVEL FUSION**

## AUTHORS NAME

Hatice Vildan Dudukcu *et al*.

## METHODOLOGY

In this work, we combine a number of weighted decision level fusion techniques with deep neural networks (DNNs) and predict blood glucose levels. By using fusion, the idea is to enhance the prediction accuracy and robustness by using multiple models. Fusion techniques combine the output of several models to make a better prediction, while the deep neural networks will search for the subtle changes in the data involving glucose rates.

**MERITS**

* Weighted decision level fusion is achieved by thoroughly integrating a number of different models, each of which predisposes to helperly contributing its own set of strengths.
* Given their qualities as complex pattern identifier deep neural networks are particularly good at identifying complex patterns in glucose data, thereby increasing the model’s ability to predict variances.
* Furthermore, this method also further enhances the robustness and dependability of the predictions, and is thus more suitable for diabetes in the real time time management.

## DEMERITS

* The deep neural networks, in particular, may be resource intensive (i.e., they require a lot of computational power).
* Another possible limitation to the model's openness to clinicians is that its intricacy might be too much to understand and change.
* low data quality or choosing the wrong fusion weights will negatively affect the model accuracy.

**2.9 AN ENSEMBLE MACHINE LEARNING APPROACH FOR PREDICTING TYPE-I DIABETES MELLITUS BASED ON LIFESTYLE INDICATORS**

## AUTHORS NAME

Shahid Mohammad Ganie & Majid Bashir Malik

## METHODOLOGY

To train the deep neural networks and use the fusion techniques, which can be computationally expensive, the model requires a large amount of computational resources. They might find the intricacies of the model too hard to understand and adjust, so that the model is not open to the clinicians that are intended to use it. On one hand, the quality of the input data and the efficient use of fusion weights also play very important roles to the model’s performance, and a bad quality input data or using the wrong fusion parameters will harm model accuracy.

**MERITS**

* Using ensemble approach to warm the dataset for prediction, it has successfully lowered possibility of overfitting that leads to high prediction accuracy and enhances the models generalizability to the different dataset.
* The inclusion of lifestyle indicators that can be used to predict Type-I diabetes form a deeper understanding of the factors that precipitate diabetes.
* Combination of the use of several models also results in a more solid and trustworthy prediction framework.

## DEMERITS

* The quality and the relevance of the lifestyle data used could affect the model performance and it may cases deliver suboptimal predictions when associated with the features of a low quality or irrelevance.
* The complexity of the method then might increase, as the method can require a lot of preprocessing as well as feature selection to guarantee that the models are trained successfully.

**2.10 SHORT-TERM PREDICTION METHOD OF BLOOD GLUCOSE BASED ON TEMPORAL MULTI-HEAD ATTENTION MECHANISM FOR DIABETIC PATIENTS**

**AUTHORS NAME**

Guanci Yang et al.

**METHODOLOGY**

In this study, temporal multi head attention is used to predict short term blood glucose by leveraging the future context. Through the use of the attention mechanisms and historical glucose data the model can focus on the most important historical data to increase the accuracy of the predictions. The attention mechanism makes the model more able to predict short term glucose variation based on temporal dependency between glucose data.

**MERITS**

* + With the help of sophisticated attention mechanisms, the model is able to focus on the most important historical data to obtain high prediction accuracy and to detect short term glucose trends.
  + This method is suitable for control of diabetes in real time, where accurate prediction of short term is needed to make changes in treatment, based on its more robust and dependable prediction model.

**DEMERITS**

* + Integrating and fitting attention mechanisms to other elements usually is difficult and takes a lot of resources to do, ultimately requiring a lot of processing power.
  + Thus, the historical glucose data is a determining factor on how well the performance of the model will perform, as it can decrease its performance based on inadequate or missing glucose data.
  + In order to optimize the data for making precise predictions, the model might require a lot preprocessing and feature selection, so the entire process would be not only more complicated, but also taking more time as well.

**2.11 A NOVEL MACHINE LEARNING APPROACH FOR DIAGNOSING DIABETES WITH A SELF-EXPLAINABLE INTERFACE**

**AUTHORS NAME**

Gangani Dharmarathne *et al.*

**METHODOLOGY**

This study builds a diabetes diagnosis machine learning model with interpretability focus. The model comes with a self explanatory interface that produces transparent and intelligible predictions. By placing this interface, the model not only obtains accurate diagnostics but also gives brief justification for the made selections, elevating the user's access and enhancing the system's credence.

**MERITS**

* + The model improves interpretability and transparency thus helping users understand the reasons for the diabetes diagnosis.
  + Thus, supporting patients and healthcare practitioners with concise justifications for choices for diagnostics enhances the user experience and fosters more trust.
  + It may be possible to include cutting edge machine learning techniques in order to provide more accurate diagnoses.

**DEMERITS**

* The overall usability and ease of explanation of the self explanatory interface depends on the complexity of the model as it becomes harder for some of the users to explain to them.
* The creation and maintenance of such an interface requires significant computational resources for which the development and operating expenses might increase.
* This could also be the case for the model and the relate performance since the training data is of poor quality.
* Not representativeffecting on the predictability and equity of the results.

**2.12 ENHANCED BLOOD GLUCOSE LEVELS PREDICTION WITH A SMARTWATCH**

**AUTHORS NAME**

Sean Pikulin *et al*

**METHODOLOGY**

To make this estimate of blood glucose levels, this study relies on use of smartwatch sensors to track physiological data: heart rate, and physical activity level. A predictive model is then applied to evaluate this smartwatch data to come up with real-time glucose predictions. Wearable technology is used within a strategy that provides users convenient, continuous, non invasive glucose monitoring.

**MERITS**

* + The integration of wearable technology allows users to have continuous glucose monitoring, thus, giving them the convenience and non invasiveness to monitor their blood sugar levels.
  + The predictive model is a helpful tool for managing diabetes since it has the ability to provide real time glucose estimates based on the physiological data collected.
  + Such method adds value for the user as it eliminates the need for finger-prick tests by allowing more frequent monitoring.

**DEMERITS**

* The smartwatch’s sensor capabilities and the accuracy level of data it records can limit the model’s opportunity to make accurate predictions.
* The idea of data security and privacy is always a concern, since the sensitive health data gets monitored continuously.

## A BRIEF REVIEW OF NEAREST NEIGHBOR ALGORITHM FOR LEARNING AND CLASSIFICATION

**AUTHORS NAME**

K. Taunk, S. De, S. Verma, A. Swetapadma

## METHODOLOGY

Most likely, the publication included a succinct summary of nearest neighbor algorithms used in learning and classification. It's possible that this review includes summarizing important ideas like distance measures, different versions of algorithms (such k-nearest neighbors), and applications across different fields. The fundamental ideas, advantages, and disadvantages of nearest neighbor techniques, as well as implementation considerations, may have been covered by the writers.

**MERITS**

This succinct overview emphasizes the Nearest Neighbor algorithm's performance in learning and classification tasks, highlighting its versatility and ease of use across multiple domains. It sheds light on its adaptability and durability, which makes it a useful instrument for data analysis and pattern detection. The paper also explores its prospective uses in several disciplines, demonstrating its versatility and broad utility**.**

## DEMERITS

The length of the document may have constraints, since it may not go into enough detail when describing algorithm intricacies, parameter tuning, or comparisons with other approaches. It could ignore current developments or important research. Credibility may be impacted by methodological flaws such as imprecise literature

selection criteria or biased interpretation. If these flaws exist, they could reduce its usefulness as an all-inclusive source on nearest neighbor techniques.

## EARLY DETECTION OF PARKINSON’S DISEASE USING DEEP LEARNING AND MACHINE LEARNING

**AUTHORS NAME**

W. Wang, J. Lee, F. Harrou, Y. Sun

## METHODOLOGY

For early Parkinson's disease identification, the study most likely combined machine learning and deep learning methods. This required preprocessing the data and using pertinent datasets to train models such as support vector machines (SVMs) and convolutional neural networks (CNNs).

**MERITS**

The method uses deep learning and machine learning to examine complicated

data patterns to demonstrate how successful these techniques are in detecting Parkinson's disease early on. The accuracy and efficiency of identifying possible illness signs prior to the start of symptoms is improved by the integration of various strategies. Proactive measures such as this one present encouraging opportunities for prompt intervention and better patient outcomes.

## DEMERITS

The paper's potential drawbacks include its ambiguous technique description, which impedes validation efforts and replication. Furthermore, the generalizability of results may be impacted if the dataset is small or lacks diversity, which could lead to an inadequate representation of real-world populations. Reliability and applicability might be jeopardized by methodological flaws like overfitting, in which models may memorize the training data instead of drawing conclusions from it. Enhancing the paper's impact and relevance in Parkinson's disease research could be achieved by addressing these limitations through the use of larger and more diverse datasets, appropriate regularization techniques, thorough validation experiments, and clearer methodology descriptions.

## PARKINSON’S DISEASE DETECTION USING MACHINE LEARNING TECHNIQUES

**AUTHORS NAME**

N. Radha, R.M. Sachin Madhavan, S. Sameera holy

## METHODOLOGY

The study most likely used machine learning methods—which include feature extraction, data collection, and algorithm training—to identify Parkinson's disease in individuals. Metrics that were appropriate, like accuracy, sensitivity, and specificity, were used to evaluate performance. Furthermore, pretreatment procedures might have been used to improve the quality of the data, and cross-validation strategies might have been used to guarantee the reliability of the outcomes.

## MERITS

This approach shows the promise for early diagnosis of Parkinson's disease by utilizing machine learning techniques to obtain accurate and reliable identification of the condition. Through the use of sophisticated algorithms, it makes it possible to find minute trends in data that point to the illness, allowing for timely intervention and individualized treatment plans. Its use emphasizes how important computational methods are to increasing Parkinson's disease diagnosis accuracy and enhancing patient treatment**.**

## DEMERITS

One of the possible drawbacks could be a methodology document that is opaque, which would make replication difficult. Furthermore, methodological problems like overfitting could jeopardize the credibility of the conclusions, and small or biased datasets could affect how broadly the findings can be applied. The significance of the work on Parkinson's disease research could be greatly increased by addressing these limitations by employing larger and more diverse datasets, producing clearer approach descriptions, and putting suitable validation techniques into practice.

.

## MULTI-MODALITY MACHINE LEARNING PREDICTING PARKINSON’S DISEASE

**AUTHORS NAME**

M.B. Makarious, H.L. Leonard, D. Vitale, et al.

## METHODOLOGY

The study most likely used a multi-modality machine learning approach, combining information from many sources including genetic markers, neuroimaging scans, and clinical evaluations. Machine learning algorithms, such as ensemble methods or deep learning architectures, were used to aggregate data and predict illness status after feature extraction techniques were applied to extract pertinent information from each modality. Through performance evaluation with relevant criteria, the generalizability and evaluated.

**MERITS**

This novel method improves the accuracy and dependability of Parkinson's

disease diagnosis by combining various data modalities with machine learning algorithms. Through the utilization of many information sources, such as imaging and clinical data, a thorough evaluation of disease risk and development can be achieved.

## DEMERITS

difficult.

* The methodological documentation is opaque, which makes repeatability
* Limited or non-representative datasets that affect the ability to generalize.
* Reliability is impacted by methodological problems including selection bias

overfitting.

* + Clinical utility may be questioned if potential confounding factors or external validity are not addressed. By addressing these issues, the paper's ability to forecast Parkinson's disease through multi-modality machine learning techniques may be improved.

## SUMMARY

This section offers a thorough overview of the literature that explores different approaches for Parkinson's disease diagnosis and prediction using artificial intelligence and machine learning techniques. Deep learning classifiers, feature selection techniques, ensemble learning algorithms, and the importance of acoustic characteristics in the diagnosis of Parkinson's disease are among the subjects discussed. The technique, possible drawbacks, and contributions made by each study are examined, providing an overview of the state of the art in Parkinson's disease detection and prediction research at the moment.

## CHAPTER 3 SYSTEM STUDY

## OVERVIEW

This study looks into the use of machine learning methods for speech characteristics-based early Parkinson's disease (PD) detection. The goal of the research is to create a non-invasive technique for voice analysis-based Parkinson's disease (PD) . The fundamental approach uses an ensemble model that combines Extra Randomized Trees (ERT) and Extreme Gradient Boosting (XG Boost). While ERT prevents overfitting by adding randomization during tree construction, XG Boost increases model robustness through gradient boosting. In comparison to single models, this ensemble technique aims to obtain more accurate PD categorization based on vocal characteristics.

To find the most pertinent vocal characteristics for accurate PD classification, feature selection strategies such as Mean Decrease in Impurity (F-MDI), Feature Permutation (F- PER), and Pearson's Correlation (F-CORR) are investigated. For improved feature selection, the study also makes use of the Butterfly Optimization Algorithm.

Metrics for performance evaluation, including F-score, accuracy, recall, and precision, are used to evaluate the suggested ensemble method. Its efficacy in early PD detection is demonstrated through comparative study against baseline models.

The findings imply that the ensemble method—especially the XG Boost and ERT combination—represents a noteworthy breakthrough in voice analysis-based early Parkinson's disease identification. This work offers insightful information for future investigations into the identification of neurodegenerative illnesses.

## BUILDING ENSEMBLE MODEL OF XGB AND ERT

In this work, we investigate the use of machine learning methods for the early detection of Parkinson's disease (PD) based on speech features. Parkinson's disease is a neurological disorder that greatly impairs balance and movement, making early discovery

crucial for successful treatment. Based on the Parkinson's Speech Dataset (PSV) and its features (jitter, shimmer, pitch, and voice break), we suggest an ensemble model that combines Extra Randomized Trees (ERT) with Extreme Gradient Boosting (XG Boost). With ERT adding randomization to reduce overfitting during tree construction and XG Boost strengthening model resilience through gradient boosting, this ensemble strategy seeks to maximize the distinct benefits of each technique. Our objective is to more effectively classify PD based on vocal characteristics by combining the strengths of both approaches.

We use metrics like accuracy, precision, recall, and F1-score on a benchmark dataset pertaining to Parkinson's disease to evaluate the model's performance. In order to achieve better feature selection, we also use advanced feature selection methods including Pearson's Correlation (F-CORR), Feature Permutation (F-PER), and Mean Decrease in Impurity (F-MDI), which are enhanced by the Butterfly Optimization Algorithm. Our study's findings not only contribute to the advancement of non-invasive techniques for early PD detection, but they also have the potential to improve patient care and quality of life by increasing diagnostic accuracy.

## SUMMARY

Using the Parkinson's Speech Dataset (PSV), the study explores the use of machine learning approaches for early detection of Parkinson's disease (PD) based on speech features. It suggests an ensemble model that combines Extra Randomized Trees (ERT) and Extreme Gradient Boosting (XG Boost) to produce PD categorization that is more accurate based on vocal traits. Along with the Butterfly Optimization Algorithm for improved feature selection, feature selection methods including F-MDI, F-PER,-FCORR are investigated. When compared to baseline models, performance evaluation parameters show how effective the ensemble approach is at identifying Parkinson's disease early on. The study represents a substantial development in non-invasive PD diagnosis techniques, and patient care through enhanced aacuracy.

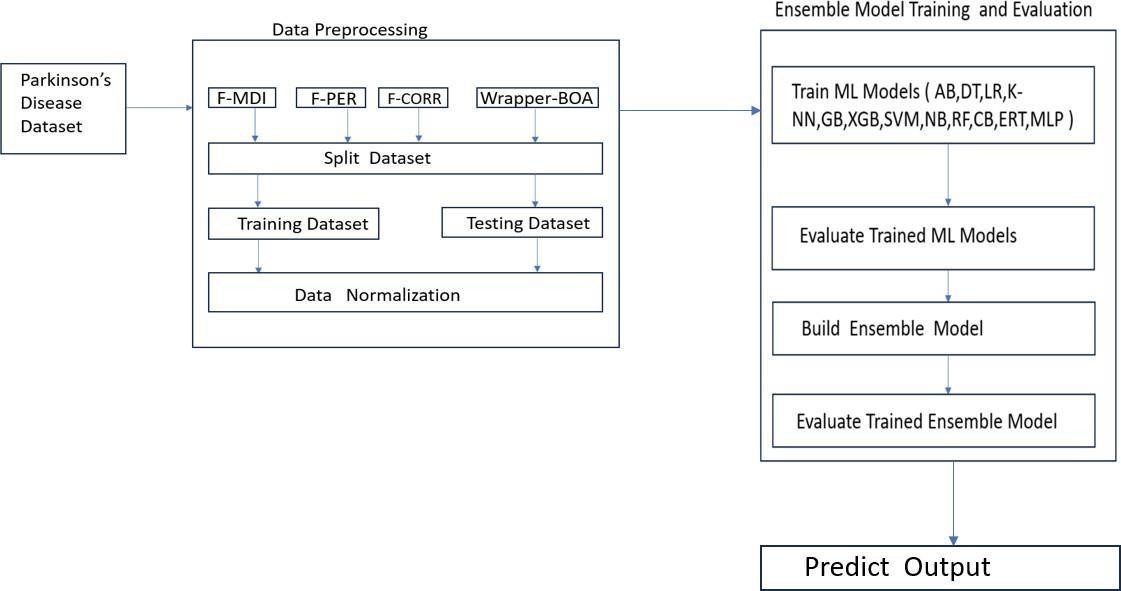
## CHAPTER 4 SYSTEM DESIGN

## OVERVIEW

The approach that has been put forth is a ground-breaking and comprehensive framework for the early detection of Parkinson's disease (PD) that integrates the cutting-edge Ensemble Machine Learning (ML) techniques with the innovative Butterfly Optimization algorithm. The ultimate goal is to use voice data, particularly from the Parkinson's Speech Dataset (PSV), to predict early symptoms of Parkinson's disease (PD) in a subtle way. The proposed system was divided into two stages.In the initial part of the data preprocessing process, attribute significance was determined using the F-MDI, F-PER, F-CORR, and BOA techniques. Seventy-five examples of the dataset with the selected features made up eighty percent of the dataset; the remaining twenty percent was designated as the test set.

The training and test sets were equalized using the Min-Max scaling technique. In the second stage, the dataset was divided into Parkinson and Not Parkinson classes. Many machine learning methods were first trained using the training dataset.

## 4.2. SYSTEM ARCHITECTURAL DESIGN

****

**Figure 4.1** Proposed System

## MODULE DESCRIPTION

The module suggests integrating Extra Randomized Trees (ERT) and Extreme Gradient Boosting (XG Boost) in an ensemble model for diagnosing Parkinson's disease based on speech features. This method makes use of vocal characteristics such as pitch, shimmer, jitter, and voice break to increase the accuracy of classification. The core of the approach is the Nearest Neighbor Boosting (NNB) strategy, which combines the k- Nearest Neighbor and Gradient Boosting approaches. The Butterfly Optimization algorithm and advanced feature selection techniques improve model performance. The results demonstrate improved performance above baseline models, indicating a major breakthrough in the early detection of Parkinson's disease.

## DATA PREPROCESSING

Preparing the data is a crucial step in machine learning. It entails converting unprocessed data into a format that machine learning algorithms can comprehend. The "Min-Max" scaling methodology was utilized in the data preparation stage of this work to normalize PD, and the FMDI, F-PER, and F-CORR processes were employed to select significant features.Parkinson’s Speech Dataset (PSV) to identify the subtle symptoms of Parkinson’s disease By using this recommended data pretreatment technique, we want to increase the quality and value of our dataset for machine learning applications. The process consists of multiple crucial parts and starts with missing data imputation (FMDI). Here, we identify and correct missing values in the dataset using advanced imputation techniques including regression and K-nearest neighbors. Following F-MDI, we move on to Feature Engineering (F-PER).

## ENSEMBLE MODEL TRAINING AND EVALUATION:

* + - 1. **Train ML Models:** A variety of machine learning models, from more complex algorithms like Gradient Boosting(GB), XGBoost(XGB), and Multilayer Perceptron(MLP),Decision Trees(DT), Logistic Regression(LR), Ada Boosting(AB), Random Forest(RF),K-NN, Support Vector Machine(SVM), Naive Baiyes(NB),Category Boosting(CB) and Extra Randomzied Trees(ERT) are used at the start of the training phase. This guarantees a thorough investigation of model functionalities and architectures.
      2. **Model-Specific Tuning:** To maximize the performance of every single model, hyperparameter optimization is a must. Methods such as grid search or random search methodically investigate the hyperparameter space in order to find the combination that produces optimal outcomes.
      3. **Cross-Validation:** Cross-validation is used to evaluate how well the models generalize to new data and prevent overfitting. Using this method, the dataset is divided into numerous folds, the models are trained on distinct subsets, and their performance is assessed over multiple data divisions.
      4. **Connect with Evaluate Trained ML Models:** Performance Evaluation: Each model is trained separately, and then its performance is assessed using a specific testing dataset. Metrics like recall, accuracy, precision, and F1-score give a thorough picture of the advantages and disadvantages of each model.
      5. **Comprehensive Report:** The results of the evaluation are compiled in a thorough report. This report facilitates model comparison, highlights the capabilities of each model, and provides guidance for the ensemble selection process.

## Build Ensemble Model:

* + - * 1. **Model Combination:** The predictions of the individual models are combined to create the ensemble model.
        2. **Weighting:** It is possible to prioritize more significant models inside the ensemble by giving individual models weights. Weights may be established in accordance with their evaluation results or domain knowledge. .
      1. **Connect with Evaluate Trained Ensemble Model:** Ensemble Evaluation: Using the same metrics as for individual models, the ensemble model is then assessed on the testing dataset in the last stage. This assessment sheds light on the ensemble’s overall capacity for prediction.Comparison with Individual Models, One can evaluate the value addition provided by merging models by contrasting the ensemble’s performance with that of the individual models. By utilizing the advantages of various models and minimizing the shortcomings of the 6 individual models, ensemble models frequently

exhibit better performance. To sum up, this all-inclusive process integrates the best features of different machine learning models by means of independent training, assessment, and then group building. The iterative procedure, which includes constant evaluation and hyperparameter adjustment, guarantees the models’ adaptability and efficacy as new data become available over time. To preserve openness and enable upcoming enhancements, the methodology is well documented..

## PREDICT OUTPUT

Through an intricate analysis of speech characteristics like jitter, shimmer, pitch, and voice break, we've developed a robust method to classify individuals into Parkinson's and Non-Parkinson's classes. By employing a combination of Extreme Gradient Boosting (XG Boost) and Extra Randomized Trees (ERT) within an ensemble model, we've significantly improved our ability to detect Parkinson's disease early on. Ultimately, we are able to classify individuals into Parkinson class and Non-Parkinson class based on the chosen subset of attributes, resulting in improved accuracy and reliability in identifying Parkinson's disease using vocal characteristics.

## SUMMARY

For the early detection of Parkinson's disease (PD), the novel Butterfly Optimization algorithm is combined with state-of-the-art Ensemble Machine Learning techniques in the suggested method. The method seeks to identify early indications of Parkinson's disease (PD) by evaluating speech features from the Parkinson's Speech Dataset (PSV). The procedure entails preprocessing the data, selecting features using the F-MDI, F-PER, and F-CORR techniques, and then training and assessing the ensemble model. With the integration of Extra Randomized Trees (ERT) and Extreme Gradient Boosting (XG Boost), the model is able to categorize individuals with more accuracy into Parkinson and Non-Parkinson classes based on vocal features. All things considered, this method offers a thorough framework for improving PD diagnosis using machine learning- based speech data analysis.

**CHAPTER 5**

**SYSTEM IMPLEMENTATION**

## OVERVIEW

Using speech data from the Parkinson's Speech Dataset (PSV), the suggested approach combines the Ensemble Machine Learning techniques with the Butterfly Optimization algorithm for early Parkinson's Disease (PD) detection. Attribute significance is ascertained and a subset of features is refined for classification through rigorous data preprocessing. Through the training and assessment of multiple machine learning models, such as Gradient Boosting, XGBoost, and Extra Randomized Trees, the framework attains improved precision and dependability in detecting subtle indications of Parkinson's disease. This emphasizes how important it is to preprocess data before using machine learning models to optimize them for PD detection.

## F-MDI ALGORITHM:

The F-MDI uses a node, which is described in algorithm 1, to calculate the feature’s importance.

Algorithm 1: F-MDI Algorithm **Input**: X: Feature matrix as input **Output**: F-MDI features

1. Set up the F-MDI using zeros (X.shape[1]);
2. Execute a loop using the Trees (T in model.estimators );
3. foreach tree in model.estimators do
4. zeros for node importance(X.shape[1]);
5. stack: [0];
6. Node id = stack.pop() when stacking;
7. while stack is not empty do
8. if tree.children left[node id] tree.children right[node id] then
9. tree.impurity[node id] = impurity reduction (tree.children left[node id], tree.impurity) + tree.impurity[tree.children right[node id]];
10. end
11. node importance += Reduction of impurities;
12. stack.append(left[node id] of tree.children);
13. stack.append(right[node id] of tree.children);
14. end
15. F-MDI += significance of node;
16. end
17. Standardize by using F-MDI /= len(model.estimators );
18. return F-MDI features

## F-PER ALGORITHM:

The following Algorithm 2 explains how the F-PER computs a score based on the target variable and related features.

Algorithm 2: F-PER Algorithm

**Input**: X scaled: Scaled feature matrix as input y: Labels for target variables

k: The desired feature count, which is 10 by default

1. F-PER Selector = SelectKBest(f classif, k = k);
2. X selected = Selector.fit transform(X scaled, y);
3. selected features = X.columns[Selector.get support()];
4. Use clf = RandomF orestClassif ier(random state = 42) to train the Random Forest Classifier;
5. clf.fit(X selected, y);
6. X scaled = X selected[:, Selector.get support() for Selector];
7. feature importance = pd.Series(index=selected features, clf.feature importances .sort values(ascending=False));
8. return significant F-PER-based features

## F-CORR ALGORITHM:

Algorithm 3 explains how to get the F-CORR score using standard deviation and co- variance.F-CORR is a standardized measure of covariance, often ranging between ”1 and 1,” and is calculated by dividing the normative variances of the two variables by their

covariance. A strong positive correlation is indicated by 1, a strong negative correlation by 1, and no correlation at all by 0.

Algorithm 3: F-CORR Algorithm

**Input**: X scaled: Scaled feature matrix as input Y : Labels for target variables

**Output**: Features with favorable F-CORR

1. Features and Enumeration:
2. features = df.drop(axis=1, ’class’);
3. features positive fcorr = [];
4. Calculate and iterate:
5. foreach feature in features.columns do
6. f corr score = np.cov(df[f eature],df[ ′ class′ ])[0,1] (np.std(df[feature])×np.std(df[’class’])) ;
7. if f corr score > 0 and f corr score ̸= 1 then
8. features positive fcorr.append(feature);
9. end
10. end

11. return features positive fcorr

## BOA ALGORITHM:

The BOA computes the objective function and determines the optimal solution through the use of an exploration phase that identifies the most significant characteristics and a local search phase that performs optimization, as described in Algorithm 4.

Algorithm 4: Butterfly Optimization Algorithm

**Input**: X: Feature matrix

y: Target variable values

feature names: List of feature names

n iterations: Total number of optimization iterations n population: Number of butterflies in the population n selected: Number of features to choose

1. Initialization:
2. Set parameters: p (switch probability), a (power exponent), and c (sensor modularity);
3. Define objective function: objective function;
4. Create a population of n population butterflies;
5. Initialize stimulus intensities I for each butterfly using the objective function;
6. Optimization Process:
7. for iteration ← 1 to n iterations do
8. foreach butterfly in the population do
9. Calculate scent based on stimulus intensity: perfume ← c × I a ;
10. if I > 0 then
11. I ← 0;
12. end
13. end
14. Identify the best butterfly with the highest objective function value;
15. foreach butterfly in the population do
16. foreach feature in butterfly do
17. Generate random number r;
18. if r < p then
19. Update feature value;
20. end
21. else
22. Select two random butterflies and perform random perturbation;
23. end
24. end
25. Return the best solution identified.

## SUMMARY

The suggested architecture uses speech data from the Parkinson's Speech Dataset (PSV) to improve the detection of Parkinson's disease (PD) by sifting through the dataset to find features that have positive correlations. F-CORR finds features that have been found to be important, and BOA uses iterative updates to optimize the model. Together, these techniques improve the accuracy of PD recognition by streamlining the processes of data preparation, feature selection, and model optimization.

## CHAPTER 6 RESULTS AND DISCUSSIONS

## DATASET DESCRIPTION

The dataset includes 756 examples relevant to computer science that are used for various classification purposes. A vast number of traits, totaling 754 properties, describe each instance. These characteristics include cross-domain and include formant frequencies that are critical to audio and speech analysis, as well as basic baseline features and intensity parameters. While variables pertaining to vocal fold features may provide additional insights into frequency characteristics, bandwidth parameters may illuminate systems involved in voice production. Additionally, there are sophisticated signal processing methods that support audio and voice analysis, such as MFCCs. In addition, wavelet features provide possible representations at various resolutions in the frequency and temporal domains. In the assigned column, each instance is linked to a class assignment, allowing for supervised learning. The absence of any missing values in the sample emphasizes

## EVALUATION METRICS

1. **Accuracy:**

A metric called "measure of overall correctness" is used to compare the percentage of properly predicted occurrences in a dataset to the total number of instances in order to assess how well a predictive model performs. Usually expressed as a percentage, this measure's higher values correspond to improved predicted accuracy.

Accuracy = TP + TN / TP + TN + FP + FN

Where TN=True Negative

TP=True Positive FN=False Negative FP=False Positive

## Precision:

The accuracy of positive predictions in relation to all positive predictions generated by the model is the main focus of precision. When a model has a high accuracy score, it means that it produces fewer false positive predictions. This makes it especially helpful in situations where reducing false positives is essential, like fraud detection or medical diagnosis.

Precision = TP / TP + FP

1. **Recall**:

No matter how many false positives the model may also predict, recall measures its capacity to identify all positive cases.

Recall = TP / TP + FN

1. **F1-Score**:

A popular performance statistic for classification tasks that offers a fair evaluation of a model's recall and precision is the F1 score.

F1 score = 2.precision.Recall / precision + recall

## OUTPUT DESCRIPTION:

## F-MDI RESULTS:

Important Features based on F-MDI Importance: ['mean\_MFCC\_12th\_coef', 'GNE\_mean', 'mean\_0th\_delta',

'mean\_delta\_log\_energy', 'mean\_MFCC\_11th\_coef', 'mean\_6th\_delta', 'mean\_9th\_delta', 'mean\_2nd\_delta', 'VFER\_SNR\_SEO', 'mean\_11th\_delta',

'mean\_MFCC\_4th\_coef', 'IMF\_SNR\_entropy', 'mean\_10th\_delta',

'minIntensity', 'IMF\_NSR\_entropy', 'mean\_8th\_delta', 'mean\_MFCC\_3rd\_coef', 'mean\_4th\_delta', 'mean\_Log\_energy', 'ddaShimmer',

'mean\_3rd\_delta', 'mean\_MFCC\_5th\_coef', 'IMF\_SNR\_SEO', 'GNE\_NSR\_SEO', 'mean\_delta\_delta\_log\_energy', 'f4', 'mean\_1st\_delta', 'VFER\_NSR\_SEO', 'VFER\_NSR\_TKEO', 'mean\_7th\_delta', 'mean\_MFCC\_9th\_coef', 'rapJitter', ‘mean\_1st\_delta\_delta', 'mean\_MFCC\_8th\_coef', 'mean\_MFCC\_10th\_coef', 'mean\_delta\_delta\_0th', 'mean\_MFCC\_2nd\_coef', 'mean\_MFCC\_1st\_coef', 'mean\_11th\_delta\_delta', 'app\_LT\_entropy\_shannon\_10\_coef', 'app\_LT\_entropy\_log\_7\_coef', 'app\_LT\_entropy\_log\_6\_coef', 'app\_LT\_entropy\_log\_5\_coef', 'app\_LT\_entropy\_log\_4\_coef', ‘app\_LT\_entropy\_log\_3\_coef', 'app\_LT\_entropy\_log\_2\_coef']

## F-PER RESULTS

Important Features based on F-PER:

std\_9th\_delta\_delta 0.164458

mean\_MFCC\_2nd\_coef 0.150051

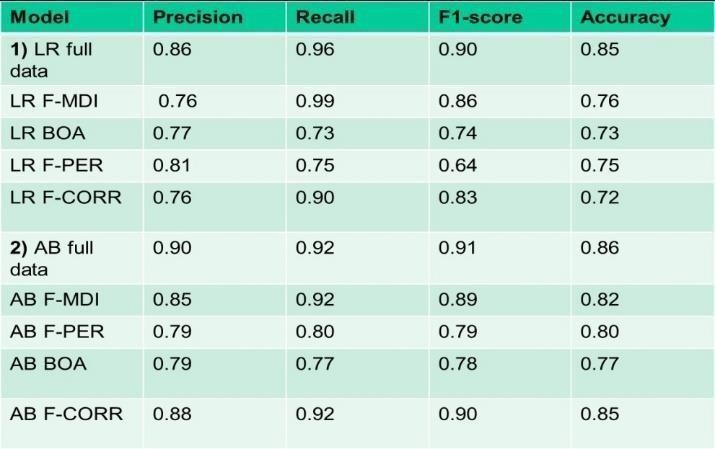
|  |  |
| --- | --- |
| tqwt\_entropy\_log\_dec\_12 | 0.117193 |
| tqwt\_stdValue\_dec\_12 | 0.111718 |
| tqwt\_minValue\_dec\_13 | 0.082850 |
| tqwt\_stdValue\_dec\_11 | 0.078645 |
| tqwt\_maxValue\_dec\_11 | 0.075963 |
| tqwt\_minValue\_dec\_11 | 0.074209 |
| tqwt\_maxValue\_dec\_12 | 0.072510 |
| tqwt\_minValue\_dec\_12  6.3.3 **BOA RESULTS** | 0.072404] |

['tqwt\_skewnessValue\_dec\_4','tqwt\_maxValue\_dec\_18','tqwt\_meanValue\_dec\_30', 'tqwt\_stdValue\_dec\_15','tqwt\_minValue\_dec\_11','tqwt\_minValue\_dec\_8', 'tqwt\_entropy\_log\_dec\_31','app\_entropy\_shannon\_7\_coef','ppq5Jitter', 'app\_LT\_entropy\_log\_6\_coef','tqwt\_stdValue\_dec\_10','GNE\_SNR\_TKEO', 'tqwt\_TKEO\_mean\_dec\_21','tqwt\_medianValue\_dec\_11','IMF\_SNR\_SEO', 'tqwt\_kurtosisValue\_dec\_12','mean\_9th\_delta\_delta','tqwt\_entropy\_log\_dec\_23',

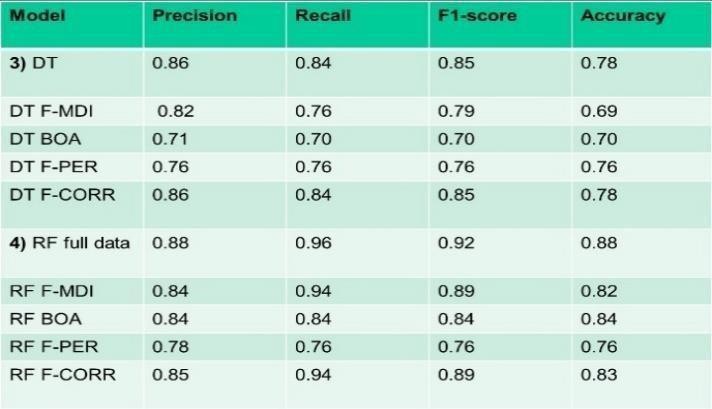
'tqwt\_kurtosisValue\_dec\_31','tqwt\_medianValue\_dec\_18','tqwt\_skewnessValue\_dec\_33',' mean\_0th\_delta','tqwt\_entropy\_log\_dec\_24','tqwt\_stdValue\_dec\_6','tqwt\_maxValue\_dec

\_10','std\_MFCC\_9th\_coef','mean\_12th\_delta','tqwt\_skewnessValue\_dec\_20','det\_LT\_entr opy\_log\_5\_coef','tqwt\_TKEO\_mean\_dec\_6','tqwt\_entropy\_log\_dec\_5', 'tqwt\_medianValue\_dec\_17', 'tqwt\_medianValue\_dec\_2']

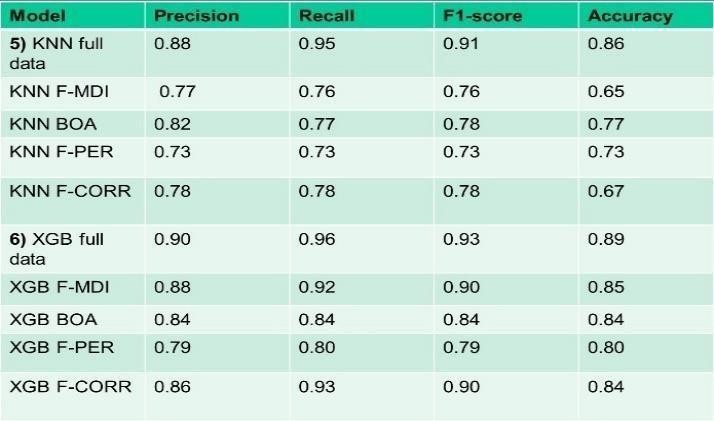
## EVALUATION METRICS RESULTS

****

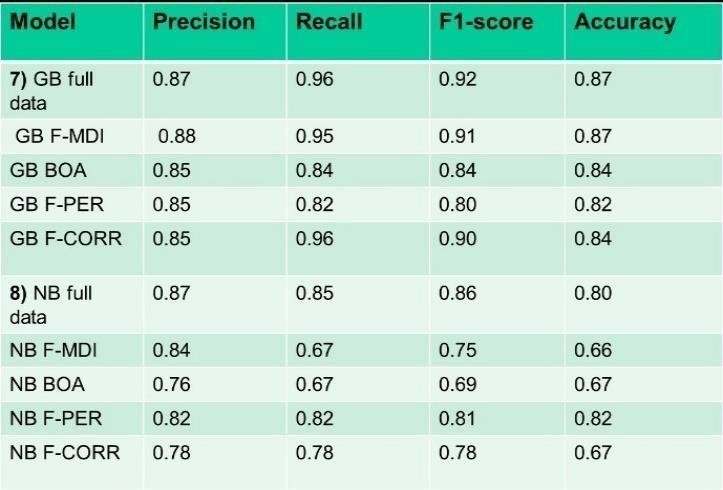
**Figure 6.1** Training of Logistic Regression and Ada Boosting



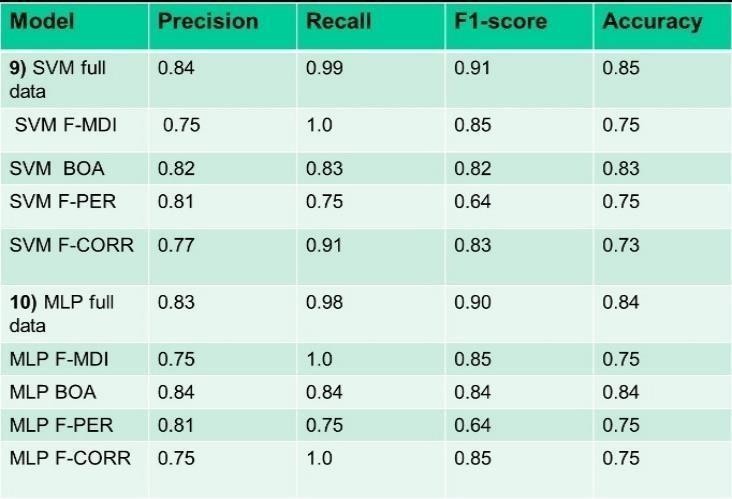
**Figure 6.2** Training of Decision Tree and Random Forest



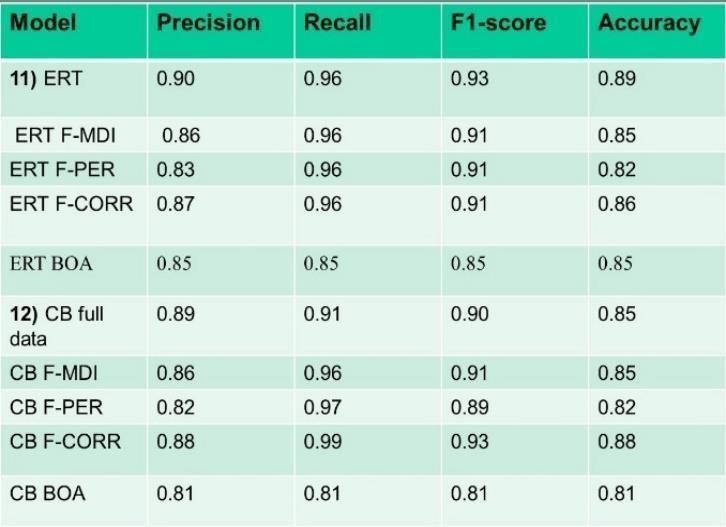
**Figure 6.3** Training of KNN and Xtreme Gradient Boosting



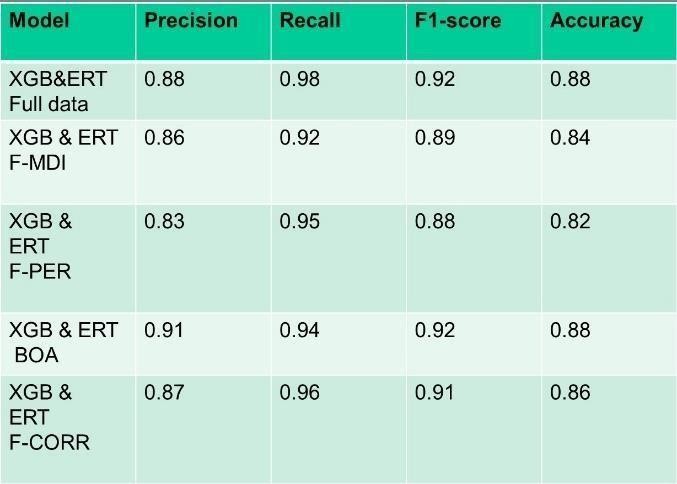
**Figure 6.4** Training of Gradient Boosting and Naive Bayes



**Figure 6.5** Training of Support Vector Machine and Multi Layer Perceptron



**Figure 6.6** Training of Extra Randomized Tree and Category Boosting



**Figure 6.7** Training of Ensembled Model XGB and ERT

According the performance parameters accuracy, recall, preci- sion, and F-score, the proposed ensembled model XGB and ERT classifier outperformed the ML models, as shown in the figure 6.4.7.The work emphasizes how crucial feature selection is to enhancing the functionality of early Parkinson’s disease detection models, which can guide the creation of more potent and successful screening instruments.

## CHAPTER 7 CONCLUSION

## CONCLUSION

To sum up, this study considerably improves the prediction of Parkinson’s disease by combining the high-accuracy output of Extra Randomized Tree and Xtreme Gradient Boosting models. The Ensemble model is incorporated into the classification process to further improve its accuracy and distinguish between individuals with Parkinson’s disease and those who are normal. The thorough methodology takes a close look at a Butterfly Optimization Algorithm, which enhances the computer technologies’ overall precision and optimality. The system’s usefulness and practical viability are demonstrated by its remarkable 88 percentage accuracy in categorizing individuals as either normal or disease-affected in real-world implementation. Personalized medical techniques and multimodal data analysis will be the main topics of future investigation.

## SOCIAL IMPACTS

**Early Diagnosis and Treatment**: ML-based prediction models have the ability to detect Parkinson's disease early on, allowing for prompt intervention and treatment. Reducing symptoms and improving quality of life are some of the advantages of early diagnosis, which may also slow the disease's progression**.**

.

**Reduced Healthcare Costs**: ML models have the potential to lower Parkinson's disease-related healthcare expenses by enabling early detection and treatments. Prompt symptom management can reduce hospitalization rates and the need for costly therapies.

**Improved Patient Care**: Predictive models powered by machine learning (ML) can help clinicians create individualized care plans that are specific to each Parkinson's patient. This can improve patient outcomes, increase patient satisfaction, and improve overall quality of care.

.

**Enhanced Research and Development**When machine learning algorithms are used to forecast Parkinson's disease, new information on the course of the condition, its risk factors, and possible biomarkers can be obtained. This knowledge can support current research initiatives to create more potent medicines, which will ultimately help those who are impacted by the illness.

.

**Empowerment and Awareness**: By giving those at risk of Parkinson's disease useful information about their health, predictive models can empower those people. Growing cognizance and understanding of the illness might promote proactive lifestyle changes and health maintenance to reduce risk factors.

**Ethical Considerations**: The ethical implications of using machine learning to forecast Parkinson's disease must be carefully considered in order to protect patient autonomy, confidentiality, and justice. These ethical issues include data privacy, informed permission, and potential biases in predicting algorithms. In general, the utilization of machine learning for Parkinson's disease prediction exhibits potential to enhance patient outcomes, curtail healthcare expenses, propel research endeavors, and enable individuals to adopt proactive health management practices.

## ECONOMIC ASPECTS

**Cost Savings**: Early health problem detection prevents the need for more complex and expensive procedures that may become essential when diseases develop over time. Instead, simpler and frequently less expensive therapies are available.

**Efficient Resource Allocation**: By pinpointing specific areas of healthcare need through targeted care allocation, resources such as medical personnel, facilities, and equipment can be optimized to provide the right level of care where it's most needed, reducing waste and improving overall efficiency.

**Time Efficency**: Accelerated Drug Development Compared to conventional approaches, machine learning may discover possible clinical trial candidates more rapidly and correctly by analyzing enormous amounts of data. This shortens the

time and expense required to introduce new medications to the market while also speeding up the development and availability of new treatments, increasing patient outcomes.

**Productivity Preservation**: Early detection and management of health issues enable individuals to maintain their ability to work and contribute to the workforce for longer periods. This reduces productivity loss due to absenteeism and disability, benefiting both individuals and the economy as a whole.

**Insurance and Policy Impacts**: Because they offer insights into risk factors and trends in population health, machine learning-derived predictive models have the potential to affect healthcare policies and insurance rates. Decisions about insurance coverage and policies may become more focused and economical as a result.

**Reduced Disability Dependency**: Early health care can stop or slow the onset of disabilities, lowering a person's need for disability payments and raising their needs.

## APPENDIX I SOFTWARE REQUIREMENTS

Programming Language used: Python Software Environment: Google Colab

Processors used: CPU (Central Processing Unit)

## APPENDIX II

Source code:

//F-MDI

import pandas as pd import numpy as np

from sklearn.ensemble import RandomForestClassifier from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# Load the dataset into a pandas DataFrame

# Replace 'your\_file.csv' with the actual file name and path df = pd.read\_csv("/content/pd\_speech\_features.csv")

# Drop any rows with missing values (if applicable) df = df.dropna()

# Separate features (X) and target variable (y)

X = df.iloc[:, :-1] # Features

y = df.iloc[:, -1] # Target variable

# Handle categorical features using one-hot encoding categorical\_columns = X.select\_dtypes(include=['object']).columns

X = pd.get\_dummies(X, columns=categorical\_columns, drop\_first=True)

# Encode the target variable le = LabelEncoder()

y = le.fit\_transform(y)

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize a Random Forest classifier rf\_model = RandomForestClassifier()

# Train the Random Forest model rf\_model.fit(X\_train, y\_train)

# Function to compute F-MDI feature importance and print important features def compute\_fmdi\_importance(model, X):

n\_trees = len(model.estimators\_)

# Placeholder for F-MDI importance rf\_fii = np.zeros(X.shape[1])

# Iterate over each tree in the Random Forest for tree in model.estimators\_:

# Placeholder for node importance node\_importance = np.zeros(X.shape[1])

# Stack to keep track of nodes stack = [0] # Start from the root while stack:

node\_id = stack.pop()

# Check if it's a non-terminal node

if tree.tree\_.children\_left[node\_id] != tree.tree\_.children\_right[node\_id]: # Calculate impurity reduction

impurity\_reduction = ( tree.tree\_.impurity[node\_id] -

tree.tree\_.impurity[tree.tree\_.children\_left[node\_id]] - tree.tree\_.impurity[tree.tree\_.children\_right[node\_id]]

)

node\_importance[node\_id] = impurity\_reduction

# Add the left and right children to the stack stack.append(tree.tree\_.children\_left[node\_id]) stack.append(tree.tree\_.children\_right[node\_id])

# Accumulate the node importance to F-MDI importance rf\_fii += node\_importance

# Normalize the F-MDI importance by the number of trees rf\_fii /= n\_trees

# Get the indices of important features (top 10%) num\_features\_to\_select = int(0.1 \* X.shape[1])

important\_feature\_indices = np.argsort(rf\_fii)[::-1][:num\_features\_to\_select]

# Print the important features

important\_features = X.columns[important\_feature\_indices] print("Important Features based on F-MDI Importance:")

print(important\_features)

# Compute F-MDI feature importance and print important features compute\_fmdi\_importance(rf\_model, X\_train)

//F-PER

import pandas as pd

from sklearn.feature\_selection import SelectKBest, f\_classif from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import accuracy\_score

# Load the dataset into a pandas DataFrame

# Replace 'dataset.csv' with the actual file path or URL of your dataset data = pd.read\_csv('/content/pd\_speech\_features.csv')

# Separate the features (X) and target variable (y)

X = data.drop('class', axis=1) # Replace 'target\_column' with the name of the target column

y = data['class']

# Standardize the features scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Perform feature selection using F-PER

selector = SelectKBest(score\_func=f\_classif, k=10) # Replace 10 with the desired number of features

X\_selected = selector.fit\_transform(X\_scaled, y) selected\_features = X.columns[selector.get\_support()]

# Train a classifier using the selected features clf = RandomForestClassifier(random\_state=42) clf.fit(X\_selected, y)

# Evaluate the classifier on the full dataset X\_selected = X\_scaled[:, selector.get\_support()] y\_pred = clf.predict(X\_selected)

accuracy = accuracy\_score(y, y\_pred) print("Accuracy:", accuracy)

# Get the important features based on F-PER feature\_importance = pd.Series(clf.feature\_importances\_, index=selected\_features).sort\_values(ascending=False) print("Important Features based on F-PER:") print(feature\_importance)

//F-CORR

import pandas as pd import numpy as np

def calculate\_fcorr(df): # Get features

features = df.drop(columns=['class'])

# Initialize an empty list to store features with positive F-CORR scores positive\_fcorr\_features = []

# Calculate F-CORR scores for each feature for feature in features.columns:

fcorr\_score = calculate\_fcorr\_score(df[feature], df['class']) if fcorr\_score > 0 and fcorr\_score != 1:

positive\_fcorr\_features.append(feature) return positive\_fcorr\_features

def calculate\_fcorr\_score(feature, target): # Covariance between feature and target

covariance = np.cov(feature, target)[0, 1]

# Standard deviation of feature std\_dev\_feature = np.std(feature)

# Standard deviation of target std\_dev\_target = np.std(target)

# Calculate F-CORR score

fcorr\_score = covariance / (std\_dev\_feature \* std\_dev\_target) return fcorr\_score

# Load the dataset into a pandas DataFrame

# Replace 'dataset.csv' with the actual file path or URL of your dataset df = pd.read\_csv("/content/pd\_speech\_features.csv")

# Assuming 'Target' is the name of your target column

# If your target is binary, you might need to encode it accordingly (e.g., 0 and 1) df['class'] = df['class'].astype(int)

# Calculate features with a positive F-CORR score without including zero positive\_fcorr\_features = calculate\_fcorr(df)

# Display the features with a positive F-CORR score without including zero print("Features with a Positive F-CORR Score (without zero):") print(positive\_fcorr\_features)

//BOA

import numpy as np import pandas as pd

from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import MinMaxScaler

# Example usage

data = pd.read\_csv('/content/pd\_speech\_features.csv') # Read your dataset here

# Drop rows with missing values data = data.dropna()

# Separate features (X) and target variable (y)

X = data.iloc[:, :-1] # Features

y = data.iloc[:, -1] # Target variable

# Extract feature names feature\_names = X.columns

# Data normalization using min-max scaling scaler = MinMaxScaler()

X\_scaled = scaler.fit\_transform(X)

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

# Butterfly Optimization Algorithm for feature selection

def butterfly\_optimization\_algorithm(X, y, feature\_names, n\_iterations=100, n\_population=50, n\_selected=50):

dim = X.shape[1] # Number of dimensions/features c = 0.1 # Adjusted Sensor modularity

a = 0.1 # Adjusted Power exponent

p = 0.1 # Adjusted Switch probability

# Define the objective function

def objective\_function(y\_true, y\_pred): """

Calculate binary cross-entropy loss between true labels and predicted labels.

Parameters:

y\_true (numpy array): True labels (ground truth), where each element is either 0 or

1.

y\_pred (numpy array): Predicted probabilities for the positive class.

Returns:

float: Binary cross-entropy loss. """

epsilon = 1e-15 # to avoid division by zero

y\_pred = np.clip(y\_pred, epsilon, 1 - epsilon) # clip to avoid log(0)

# Calculate binary cross-entropy loss

loss = -(y\_true \* np.log(y\_pred) + (1 - y\_true) \* np.log(1 - y\_pred)) return np.mean(loss)

# Initialize population

np.random.seed(42) # Setting random seed for reproducibility population = np.random.rand(n\_population, dim)

# Calculate initial stimulus intensities

Is = [objective\_function(y, np.sum(x)) for x, y in zip(X, y)]

# Main optimization loop for \_ in range(n\_iterations):

for i in range(n\_population):

I = Is[i] # Stimulus intensity determined by the objective function

# Calculate fragrance for each butterfly if I >= 0:

fragrance = c \* I\*\*a else:

fragrance = 0

# Find best butterfly

best\_butterfly = population[np.argmax([objective\_function(y, np.sum(bf)) for bf, y in zip(population, y)])]

for j in range(dim):

r = np.random.rand() if r < p:

population[i][j] = population[i][j] + (r\*\*2 \* best\_butterfly[j] - population[i][j]) \* fragrance if fragrance >= 0 else 0

else:

j\_rand = np.random.randint(0, n\_population) j\_rand2 = np.random.randint(0, n\_population)

population[i][j] = population[i][j] + (r\*\*2 \* best\_butterfly[j] - population[i][j]) \* fragrance if fragrance >= 0 else 0

# Update value of a (if needed)

# Output: Print the selected most important features

best\_solution = population[np.argmax([objective\_function(y, np.sum(bf)) for bf, y in zip(population, y)])]

sorted\_indices = np.argsort(best\_solution)[::-1] # Sort indices in descending order of importance

selected\_features\_indices = sorted\_indices[:n\_selected] # Select top n features

selected\_features = [feature\_names[i] for i in selected\_features\_indices] print("Selected most important features:", selected\_features)

# Perform feature selection using Butterfly Optimization Algorithm

butterfly\_optimization\_algorithm(X\_train, y\_train, feature\_names)

//TRAINING AND TESTING FULL DATA

import pandas as pd

from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import MinMaxScaler from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier,

GradientBoostingClassifier, ExtraTreesClassifier, BaggingClassifier from sklearn.tree import DecisionTreeClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.neighbors import KNeighborsClassifier from xgboost import XGBClassifier

from sklearn.svm import SVC

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import precision\_score, recall\_score, f1\_score, accuracy\_score

# Load the dataset into a pandas DataFrame

# Replace 'dataset.csv' with the actual file path or URL of your dataset df = pd.read\_csv('/content/pd\_speech\_features.csv')

# Assuming 'Target' is the name of your target column

# If your target is binary, you might need to encode it accordingly (e.g., 0 and 1) df['class'] = df['class'].astype(int)

# Separate features (X) and target variable (y)

X = df.drop(columns=['class']) # Features y = df['class'] # Target variable

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Perform data normalization on the training set and testing set using Min-Max scaling scaler = MinMaxScaler()

X\_train\_normalized = scaler.fit\_transform(X\_train) X\_test\_normalized = scaler.transform(X\_test)

# Train several machine learning models on the normalized training set models = {

"Logistic Regression": LogisticRegression(max\_iter=1000), "AdaBoost": AdaBoostClassifier(),

"Decision Tree": DecisionTreeClassifier(), "Random Forest": RandomForestClassifier(), "K-Nearest Neighbor": KNeighborsClassifier(), "eXtreme Gradient Boosting": XGBClassifier(),

"Gradient Boosting": GradientBoostingClassifier(), "Naïve Bayes": GaussianNB(),

"Support Vector Machine": SVC(), "Multilayer Perceptron": MLPClassifier(),

"Extra Randomized Trees": ExtraTreesClassifier(), "Category Boosting": BaggingClassifier()

}

# Dictionary to store evaluation metrics for each model metrics\_dict = {}

for name, model in models.items(): model.fit(X\_train\_normalized, y\_train) y\_pred = model.predict(X\_test\_normalized)

# Calculate evaluation metrics

precision = precision\_score(y\_test, y\_pred) recall = recall\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred)

accuracy = accuracy\_score(y\_test, y\_pred)

# Store metrics in the dictionary metrics\_dict[name] = {

"Precision": precision, "Recall": recall,

"F1": f1,

"Accuracy": accuracy

}

# Print metrics for each model print(f"{name} Metrics:")

print(f" Precision: {precision:.2f}") print(f" Recall: {recall:.2f}")

print(f" F1-Score: {f1:.2f}") print(f" Accuracy: {accuracy:.2f}") print()

# Select the top two models based on accuracy

best\_models = sorted(metrics\_dict.items(), key=lambda x: x[1]["Accuracy"], reverse=True)[:2]

print("Best Performing Models based on Accuracy:") for name, metrics in best\_models:

print(f"{name}: Accuracy = {metrics['Accuracy']:.2f}, Precision =

{metrics['Precision']:.2f}, Recall = {metrics['Recall']:.2f}, F1-Score =

{metrics['F1']:.2f}")

// F-MDI TRAINED ML MODEL

# Import necessary libraries import pandas as pd

from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier, ExtraTreesClassifier, GradientBoostingClassifier

from sklearn.tree import DecisionTreeClassifier from sklearn.neighbors import KNeighborsClassifier from sklearn.naive\_bayes import GaussianNB

from sklearn.svm import SVC

from sklearn.neural\_network import MLPClassifier from xgboost import XGBClassifier

from catboost import CatBoostClassifier

from sklearn.metrics import classification\_report, accuracy\_score

# Load the dataset into a pandas DataFrame

# Replace 'dataset.csv' with the actual file name and path data = pd.read\_csv('/content/pd\_speech\_features.csv')

# Split the data into features and target variable # Add all the feature columns

X = data[['mean\_MFCC\_12th\_coef', 'GNE\_mean', 'mean\_0th\_delta', 'mean\_delta\_log\_energy', 'mean\_MFCC\_11th\_coef', 'mean\_6th\_delta', 'mean\_9th\_delta', 'mean\_2nd\_delta', 'VFER\_SNR\_SEO', 'mean\_11th\_delta', 'mean\_MFCC\_4th\_coef', 'IMF\_SNR\_entropy', 'mean\_10th\_delta', 'minIntensity', 'IMF\_NSR\_entropy', 'mean\_8th\_delta', 'mean\_MFCC\_3rd\_coef', 'mean\_4th\_delta', 'mean\_Log\_energy', 'ddaShimmer',

'mean\_3rd\_delta', 'mean\_MFCC\_5th\_coef', 'IMF\_SNR\_SEO', 'GNE\_NSR\_SEO', 'mean\_11th\_delta\_delta', 'app\_LT\_entropy\_shannon\_10\_coef', 'app\_LT\_entropy\_log\_7\_coef', 'app\_LT\_entropy\_log\_6\_coef', 'app\_LT\_entropy\_log\_5\_coef', 'app\_LT\_entropy\_log\_4\_coef', 'app\_LT\_entropy\_log\_3\_coef', 'app\_LT\_entropy\_log\_2\_coef']]

y = data['class'] # Replace 'target\_column' with the actual target column name

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize the models models = {

'Logistic Regression': LogisticRegression(), 'AdaBoost': AdaBoostClassifier(), 'Decision Tree': DecisionTreeClassifier(), 'Random Forest': RandomForestClassifier(),

'K-Nearest Neighbor': KNeighborsClassifier(), 'eXtreme Gradient Boosting': XGBClassifier(), 'Gradient Boosting': GradientBoostingClassifier(), 'Naïve Bayes': GaussianNB(),

'Support Vector Machine': SVC(), 'Multilayer Perceptron': MLPClassifier(),

'Extra Randomized Trees': ExtraTreesClassifier(), 'Category Boosting': CatBoostClassifier(verbose=False)

}

# Train the models and evaluate their performance for name, model in models.items():

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

report = classification\_report(y\_test, y\_pred, zero\_division=1) # Add zero\_division parameter

print(f"Model: {name}\nAccuracy: {accuracy}\nClassification Report:\n{report}\n")

//F-PER TRAINED ML MODEL

# Import necessary libraries import pandas as pd

from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier,

ExtraTreesClassifier, GradientBoostingClassifier from sklearn.tree import DecisionTreeClassifier from sklearn.neighbors import KNeighborsClassifier from sklearn.naive\_bayes import GaussianNB

from sklearn.svm import SVC

from sklearn.neural\_network import MLPClassifier from xgboost import XGBClassifier

from catboost import CatBoostClassifier

from sklearn.metrics import classification\_report, accuracy\_score

# Load the dataset into a pandas DataFrame

# Replace 'dataset.csv' with the actual file name and path data = pd.read\_csv('/content/pd\_speech\_features.csv')

# Select the specified features

selected\_features = ['std\_9th\_delta\_delta', 'mean\_MFCC\_2nd\_coef', 'tqwt\_entropy\_log\_dec\_12', 'tqwt\_stdValue\_dec\_12', 'tqwt\_minValue\_dec\_13', 'tqwt\_stdValue\_dec\_11', 'tqwt\_maxValue\_dec\_11', 'tqwt\_minValue\_dec\_11', 'tqwt\_maxValue\_dec\_12', 'tqwt\_minValue\_dec\_12']

X = data[selected\_features]

y = data['class'] # Replace 'target\_column' with the actual target column name

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize the models models = {

'Logistic Regression': LogisticRegression(), 'AdaBoost': AdaBoostClassifier(), 'Decision Tree': DecisionTreeClassifier(), 'Random Forest': RandomForestClassifier(),

'K-Nearest Neighbor': KNeighborsClassifier(), 'eXtreme Gradient Boosting': XGBClassifier(), 'Gradient Boosting': GradientBoostingClassifier(), 'Naïve Bayes': GaussianNB(),

'Support Vector Machine': SVC(),

'Multilayer Perceptron': MLPClassifier(),

'Extra Randomized Trees': ExtraTreesClassifier(), 'Category Boosting': CatBoostClassifier(verbose=False)

}

for name, model in models.items(): model.fit(X\_train, y\_train) y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

report = classification\_report(y\_test, y\_pred, zero\_division=1) # Add zero\_division parameter

print(f"Model: {name}\nAccuracy: {accuracy}\nClassification Report:\n{report}\n")

//F-CORR TRAINED ML MODEL

# Import necessary libraries import pandas as pd

from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier,

ExtraTreesClassifier, GradientBoostingClassifier from sklearn.tree import DecisionTreeClassifier from sklearn.neighbors import KNeighborsClassifier from sklearn.naive\_bayes import GaussianNB

from sklearn.svm import SVC

from sklearn.neural\_network import MLPClassifier from xgboost import XGBClassifier

from catboost import CatBoostClassifier

from sklearn.metrics import classification\_report, accuracy\_score

# Load the dataset into a pandas DataFrame

# Replace 'dataset.csv' with the actual file name and path data = pd.read\_csv('/content/pd\_speech\_features.csv')

# Select the specified features

selected\_features = ['gender', 'DFA', 'RPDE', 'meanPeriodPulses', 'stdDevPeriodPulses', 'locPctJitter', 'locAbsJitter', 'rapJitter', 'ppq5Jitter', 'ddpJitter', 'locShimmer', 'mean\_MFCC\_2nd\_coef', 'mean\_MFCC\_3rd\_coef', 'mean\_MFCC\_4th\_coef', 'mean\_MFCC\_5th\_coef', 'mean\_MFCC\_7th\_coef', 'mean\_MFCC\_8th\_coef', 'mean\_MFCC\_9th\_coef', 'mean\_MFCC\_11th\_coef', 'mean\_MFCC\_12th\_coef', 'mean\_4th\_delta', 'mean\_5th\_delta', 'mean\_6th\_delta', 'mean\_7th\_delta', 'mean\_10th\_delta', 'mean\_11th\_delta', 'mean\_12th\_delta', 'mean\_1st\_delta\_delta', 'mean\_2nd\_delta\_delta', 'mean\_4th\_delta\_delta', 'mean\_5th\_delta\_delta', 'mean\_6th\_delta\_delta', 'mean\_9th\_delta\_delta', 'mean\_10th\_delta\_delta', 'det\_LT\_TKEO\_std\_3\_coef', 'det\_LT\_TKEO\_std\_4\_coef', 'det\_LT\_TKEO\_std\_5\_coef', 'det\_LT\_TKEO\_std\_6\_coef', 'det\_LT\_TKEO\_std\_7\_coef', 'det\_LT\_TKEO\_std\_8\_coef', 'det\_LT\_TKEO\_std\_9\_coef', 'det\_LT\_TKEO\_std\_10\_coef', 'tqwt\_skewnessValue\_dec\_21', 'tqwt\_skewnessValue\_dec\_24', 'tqwt\_skewnessValue\_dec\_25', 'tqwt\_skewnessValue\_dec\_27', 'tqwt\_kurtosisValue\_dec\_22', 'tqwt\_kurtosisValue\_dec\_23', 'tqwt\_kurtosisValue\_dec\_32',

'tqwt\_kurtosisValue\_dec\_33', 'tqwt\_kurtosisValue\_dec\_34', 'tqwt\_kurtosisValue\_dec\_35', 'tqwt\_kurtosisValue\_dec\_36'] # Add all the specified features

X = data[selected\_features]

y = data['class'] # Replace 'target\_column' with the actual target column name

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize the models with increased max\_iter for Logistic Regression and zero\_division parameter for classification\_report

logistic\_model = LogisticRegression(max\_iter=1000)

models = {

'Logistic Regression': logistic\_model, 'AdaBoost': AdaBoostClassifier(), 'Decision Tree': DecisionTreeClassifier(),

'Random Forest': RandomForestClassifier(), 'K-Nearest Neighbor': KNeighborsClassifier(),

'eXtreme Gradient Boosting': XGBClassifier(), 'Gradient Boosting': GradientBoostingClassifier(), 'Naïve Bayes': GaussianNB(),

'Support Vector Machine': SVC(), 'Multilayer Perceptron': MLPClassifier(),

'Extra Randomized Trees': ExtraTreesClassifier(), 'Category Boosting': CatBoostClassifier(verbose=False)

}

# Train the models and evaluate their performance for name, model in models.items():

if name == 'Logistic Regression':

model.max\_iter = 1000 # Increase max\_iter for Logistic Regression model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

report = classification\_report(y\_test, y\_pred, zero\_division=1) # Add zero\_division parameter

print(f"Model: {name}\nAccuracy: {accuracy}\nClassification Report:\n{report}\n")

// BOA TRAINED ML MODEL

import pandas as pd

from sklearn.model\_selection import train\_test\_split, GridSearchCV from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier,

ExtraTreesClassifier, GradientBoostingClassifier from sklearn.tree import DecisionTreeClassifier from sklearn.neighbors import KNeighborsClassifier from sklearn.svm import SVC

from sklearn.neural\_network import MLPClassifier from xgboost import XGBClassifier

from catboost import CatBoostClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score from imblearn.over\_sampling import SMOTE

from sklearn.preprocessing import StandardScaler

from sklearn.feature\_selection import SelectFromModel from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer

# Load the dataset into a pandas DataFrame

data = pd.read\_csv('/content/pd\_speech\_features.csv')

# Handle missing values (mean imputation) data.fillna(data.mean(), inplace=True)

# Select the specified features

selected\_features = ['tqwt\_skewnessValue\_dec\_4', 'tqwt\_maxValue\_dec\_18', 'tqwt\_meanValue\_dec\_30', 'tqwt\_stdValue\_dec\_15', 'tqwt\_minValue\_dec\_11', 'tqwt\_minValue\_dec\_8', 'tqwt\_entropy\_log\_dec\_31', 'app\_entropy\_shannon\_7\_coef', 'ppq5Jitter', 'app\_LT\_entropy\_log\_6\_coef', 'tqwt\_stdValue\_dec\_10', 'GNE\_SNR\_TKEO', 'tqwt\_TKEO\_mean\_dec\_21', 'tqwt\_medianValue\_dec\_11', 'IMF\_SNR\_SEO', 'tqwt\_kurtosisValue\_dec\_12', 'mean\_9th\_delta\_delta', 'tqwt\_entropy\_log\_dec\_23', 'tqwt\_kurtosisValue\_dec\_31', 'tqwt\_medianValue\_dec\_18', 'tqwt\_skewnessValue\_dec\_33', 'mean\_0th\_delta', 'tqwt\_entropy\_log\_dec\_24', 'tqwt\_stdValue\_dec\_6', 'det\_LT\_TKEO\_std\_7\_coef', 'tqwt\_entropy\_shannon\_dec\_22', 'tqwt\_energy\_dec\_22', 'tqwt\_meanValue\_dec\_4', 'tqwt\_entropy\_log\_dec\_36', 'tqwt\_TKEO\_mean\_dec\_7', 'det\_TKEO\_std\_4\_coef', 'tqwt\_TKEO\_std\_dec\_15', 'ddpJitter', 'Ed\_2\_coef', 'tqwt\_TKEO\_std\_dec\_36', 'tqwt\_meanValue\_dec\_32', 'std\_0th\_delta', 'tqwt\_entropy\_log\_dec\_6', 'f3', 'app\_entropy\_shannon\_5\_coef', 'IMF\_NSR\_entropy', 'tqwt\_maxValue\_dec\_10', 'std\_MFCC\_9th\_coef', 'mean\_12th\_delta', 'tqwt\_skewnessValue\_dec\_20', 'det\_LT\_entropy\_log\_5\_coef', 'tqwt\_TKEO\_mean\_dec\_6', 'tqwt\_entropy\_log\_dec\_5', 'tqwt\_medianValue\_dec\_17', 'tqwt\_medianValue\_dec\_2']

X = data[selected\_features] y = data['class']

# Encode target variable to contain discrete classes (0s and 1s) y = y.astype(int)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Feature scaling

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train) X\_test\_scaled = scaler.transform(X\_test)

# Handle class imbalance using SMOTE smote = SMOTE(random\_state=42)

X\_train\_balanced, y\_train\_balanced = smote.fit\_resample(X\_train\_scaled, y\_train)

# Define models models = {

'Logistic Regression': LogisticRegression(), 'AdaBoost': AdaBoostClassifier(), 'Decision Tree': DecisionTreeClassifier(), 'Random Forest': RandomForestClassifier(),

'K-Nearest Neighbor': KNeighborsClassifier(), 'Gradient Boosting': GradientBoostingClassifier(), 'Support Vector Machine': SVC(),

'Multilayer Perceptron': MLPClassifier(),

'Extra Randomized Trees': ExtraTreesClassifier(), 'eXtreme Gradient Boosting': XGBClassifier(), 'Category Boosting': CatBoostClassifier(verbose=False), 'Naïve Bayes': GaussianNB()

}

# Hyperparameter grids for GridSearchCV param\_grids = {

'Logistic Regression': {'clf max\_iter': [100, 500, 1000], 'clf C': [0.01, 0.1, 1.0]},

'AdaBoost': {'clf n\_estimators': [50, 100, 200], 'clf learning\_rate': [0.1, 0.5, 1.0]},

'Decision Tree': {'clf max\_depth': [None, 5, 10], 'clf min\_samples\_split': [2, 5, 10]},

'Random Forest': {'clf n\_estimators': [50, 100, 200], 'clf max\_depth': [None, 5,

10]},

'K-Nearest Neighbor': {'clfn\_neighbors': [3, 5, 10]},

'Gradient Boosting': {'clf n\_estimators': [50, 100, 200], 'clf learning\_rate': [0.1, 0.5,

1.0], 'clfmax\_depth': [3, 5, 10]},

'Support Vector Machine': {'clf C': [0.1, 1, 10], 'clf gamma': ['scale', 'auto']}, 'Multilayer Perceptron': {'clfhidden\_layer\_sizes': [(100,), (50, 100, 50), (50,)],

'clfalpha': [0.0001, 0.001, 0.01], 'clfmax\_iter': [500, 1000, 2000]}, # Adjusted max\_iter values

'Extra Randomized Trees': {'clf n\_estimators': [50, 100, 200], 'clf max\_depth':

[None, 5, 10]},

'eXtreme Gradient Boosting': {'clf n\_estimators': [50, 100, 200], 'clf learning\_rate':

[0.1, 0.5, 1.0], 'clfmax\_depth': [3, 5, 10]},

'Category Boosting': {'clf iterations': [50, 100, 200], 'clf learning\_rate': [0.1, 0.5,

1.0]},

'Naïve Bayes': {} # No hyperparameters to tune for Naïve Bayes

}

# Fit and evaluate each model

for name, model in models.items(): pipe = Pipeline([

('clf', model)

])

# Add hyperparameter tuning if available

if name in param\_grids:

grid\_search = GridSearchCV(pipe, param\_grids[name], cv=5, scoring='accuracy') grid\_search.fit(X\_train\_balanced, y\_train\_balanced)

model = grid\_search.best\_estimator\_.named\_steps['clf']

# Fit the model

model.fit(X\_train\_balanced, y\_train\_balanced)

# Predictions

y\_pred = model.predict(X\_test\_scaled)

# Calculate classification metrics

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred, average='weighted', zero\_division=1) #

Adjusted to handle zero division

recall = recall\_score(y\_test, y\_pred, average='weighted') f1 = f1\_score(y\_test, y\_pred, average='weighted')

# Print the metrics print(f"Model: {name}") print(f"Accuracy: {accuracy}") print(f"Precision: {precision}") print(f"Recall: {recall}") print(f"F1-score: {f1}")

print()

// ENSEMBLE FULL DATA

import pandas as pd

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.ensemble import GradientBoostingClassifier, ExtraTreesClassifier,

VotingClassifier

from sklearn.metrics import precision\_score, recall\_score, f1\_score, accuracy\_score, roc\_auc\_score

from sklearn.preprocessing import StandardScaler import xgboost as xgb

# 1. Read dataset

psv\_data = pd.read\_csv("/content/pd\_speech\_features.csv") # Replace with your actual file path

# 2. Split dataset into training and testing sets

tr\_psv, tst\_psv = train\_test\_split(psv\_data, test\_size=0.2, random\_state=42)

# 3. Split input and output variables

tr\_x, tr\_y = tr\_psv.drop("class", axis=1), tr\_psv["class"] # Replace "target\_column" with your actual target column name

tst\_x, tst\_y = tst\_psv.drop("class", axis=1), tst\_psv["class"]

# 4. Compute feature importance (using XGBoost for enhanced feature importance)

xgb\_model = xgb.XGBClassifier() xgb\_model.fit(tr\_x, tr\_y)

feature\_importances = pd.DataFrame({"feature": tr\_x.columns, "importance": xgb\_model.feature\_importances\_})

# 5. Normalize features scaler = StandardScaler()

tr\_x = scaler.fit\_transform(tr\_x) tst\_x = scaler.transform(tst\_x)

# 6. Hyperparameter tuning (example using GridSearchCV) xgb\_param\_grid = {

"n\_estimators": [100, 200, 300],

"max\_depth": [3, 5, 7],

# ... other XGBoost hyperparameters

}

et\_param\_grid = {

"n\_estimators": [100, 200, 300],

"max\_depth": [5, 8, 10],

# ... other ExtraTrees hyperparameters

}

xgb\_grid\_search = GridSearchCV(xgb.XGBClassifier(), xgb\_param\_grid, cv=5) xgb\_grid\_search.fit(tr\_x, tr\_y)

best\_xgb\_params = xgb\_grid\_search.best\_params\_

et\_grid\_search = GridSearchCV(ExtraTreesClassifier(), et\_param\_grid, cv=5) et\_grid\_search.fit(tr\_x, tr\_y)

best\_et\_params = et\_grid\_search.best\_params\_

# Create base models with best hyperparameters xgb\_model = xgb.XGBClassifier(\*\*best\_xgb\_params) et\_model = ExtraTreesClassifier(\*\*best\_et\_params)

# 7. Create ensemble model with voting

ensemble\_model = VotingClassifier(estimators=[("xgb", xgb\_model), ("et", et\_model)], voting="soft")

# 8. Train ensemble model ensemble\_model.fit(tr\_x, tr\_y)

# 9. Predict target class

predictions = ensemble\_model.predict(tst\_x)

# 10. Compute performance metrics

precision = precision\_score(tst\_y, predictions) recall = recall\_score(tst\_y, predictions)

f1 = f1\_score(tst\_y, predictions)

accuracy = accuracy\_score(tst\_y, predictions) roc\_auc = roc\_auc\_score(tst\_y, predictions)

print("Precision:", precision) print("Recall:", recall) print("F1-score:", f1) print("Accuracy:", accuracy)

print("Area under ROC curve:", roc\_auc)

// ENSEMBLE F- MDI

import pandas as pd

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.ensemble import GradientBoostingClassifier, ExtraTreesClassifier,

VotingClassifier

from sklearn.metrics import precision\_score, recall\_score, f1\_score, accuracy\_score, roc\_auc\_score

from sklearn.preprocessing import StandardScaler import xgboost as xgb

# 1. Read dataset

psv\_data = pd.read\_csv("/content/pd\_speech\_features.csv") # Replace with your actual file path

# 2. Split dataset into training and testing sets

tr\_psv, tst\_psv = train\_test\_split(psv\_data, test\_size=0.2, random\_state=42)

# 3. Split input and output variables

features = ['mean\_MFCC\_12th\_coef', 'GNE\_mean', 'mean\_0th\_delta', 'mean\_delta\_log\_energy', 'mean\_MFCC\_11th\_coef', 'mean\_6th\_delta', 'mean\_9th\_delta', 'mean\_2nd\_delta', 'VFER\_SNR\_SEO', 'mean\_11th\_delta', 'mean\_MFCC\_4th\_coef', 'IMF\_SNR\_entropy', 'mean\_10th\_delta', 'minIntensity', 'IMF\_NSR\_entropy', 'mean\_8th\_delta', 'mean\_MFCC\_3rd\_coef', 'mean\_4th\_delta', 'mean\_Log\_energy', 'ddaShimmer',

'mean\_3rd\_delta', 'mean\_MFCC\_5th\_coef', 'IMF\_SNR\_SEO', 'GNE\_NSR\_SEO', 'mean\_delta\_delta\_log\_energy', 'f4', 'mean\_1st\_delta', 'VFER\_NSR\_SEO', 'locAbsJitter', 'IMF\_NSR\_SEO', 'mean\_5th\_delta\_delta', 'VFER\_SNR\_TKEO', 'VFER\_entropy', 'mean\_5th\_delta', 'IMF\_NSR\_TKEO', 'mean\_12th\_delta', 'VFER\_NSR\_TKEO', 'mean\_7th\_delta', 'mean\_MFCC\_9th\_coef', 'rapJitter', 'mean\_1st\_delta\_delta', 'mean\_MFCC\_8th\_coef', 'mean\_MFCC\_10th\_coef', 'mean\_delta\_delta\_0th', 'mean\_MFCC\_2nd\_coef', 'mean\_MFCC\_1st\_coef',

'f3', 'mean\_10th\_delta\_delta', 'f1', 'meanHarmToNoiseHarmonicity', 'mean\_6th\_delta\_delta', 'std\_Log\_energy', 'mean\_4th\_delta\_delta', 'mean\_MFCC\_6th\_coef', 'mean\_3rd\_delta\_delta', 'mean\_MFCC\_0th\_coef', 'mean\_2nd\_delta\_delta', 'stdDevPeriodPulses', 'mean\_9th\_delta\_delta', 'std\_MFCC\_3rd\_coef', 'mean\_8th\_delta\_delta', 'meanNoiseToHarmHarmonicity', 'GNE\_NSR\_TKEO', 'apq11Shimmer', 'VFER\_std', 'meanPeriodPulses', 'std\_MFCC\_1st\_coef', 'mean\_11th\_delta\_delta', 'app\_LT\_entropy\_shannon\_10\_coef', 'app\_LT\_entropy\_log\_7\_coef', 'app\_LT\_entropy\_log\_6\_coef',

'app\_LT\_entropy\_log\_5\_coef', 'app\_LT\_entropy\_log\_4\_coef', 'app\_LT\_entropy\_log\_3\_coef', 'app\_LT\_entropy\_log\_2\_coef'] # Add all the specified

features

tr\_x, tr\_y = tr\_psv[features], tr\_psv["class"] # Replace "target\_column" with your actual target column name

tst\_x, tst\_y = tst\_psv[features], tst\_psv["class"]

# 4. Normalize features scaler = StandardScaler()

tr\_x = scaler.fit\_transform(tr\_x) tst\_x = scaler.transform(tst\_x)

# 5. Hyperparameter tuning (example using GridSearchCV) xgb\_param\_grid = {

"n\_estimators": [100, 200, 300],

"max\_depth": [3, 5, 7],

# ... other XGBoost hyperparameters

}

et\_param\_grid = {

"n\_estimators": [100, 200, 300],

"max\_depth": [5, 8, 10],

# ... other ExtraTrees hyperparameters

}

xgb\_grid\_search = GridSearchCV(xgb.XGBClassifier(), xgb\_param\_grid, cv=5) xgb\_grid\_search.fit(tr\_x, tr\_y)

best\_xgb\_params = xgb\_grid\_search.best\_params\_

et\_grid\_search = GridSearchCV(ExtraTreesClassifier(), et\_param\_grid, cv=5) et\_grid\_search.fit(tr\_x, tr\_y)

best\_et\_params = et\_grid\_search.best\_params\_

# Create base models with best hyperparameters xgb\_model = xgb.XGBClassifier(\*\*best\_xgb\_params) et\_model = ExtraTreesClassifier(\*\*best\_et\_params)

# 6. Create ensemble model with voting

ensemble\_model = VotingClassifier(estimators=[("xgb", xgb\_model), ("et", et\_model)], voting="soft")

# 7. Train ensemble model ensemble\_model.fit(tr\_x, tr\_y)

# 8. Predict target class

predictions = ensemble\_model.predict(tst\_x)

# 9. Compute performance metrics

precision = precision\_score(tst\_y, predictions) recall = recall\_score(tst\_y, predictions)

f1 = f1\_score(tst\_y, predictions)

accuracy = accuracy\_score(tst\_y, predictions) roc\_auc = roc\_auc\_score(tst\_y, predictions)

print("Precision:", precision) print("Recall:", recall) print("F1-score:", f1) print("Accuracy:", accuracy)

print("Area under ROC curve:", roc\_auc)

// ENSEMBLE F-PER

import pandas as pd

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.ensemble import GradientBoostingClassifier, ExtraTreesClassifier,

VotingClassifier

from sklearn.metrics import precision\_score, recall\_score, f1\_score, accuracy\_score, roc\_auc\_score

from sklearn.preprocessing import StandardScaler import xgboost as xgb

# 1. Read dataset

psv\_data = pd.read\_csv("/content/pd\_speech\_features.csv") # Replace with your actual file path

# 2. Split dataset into training and testing sets

tr\_psv, tst\_psv = train\_test\_split(psv\_data, test\_size=0.2, random\_state=42)

# 3. Split input and output variables

features = ['std\_9th\_delta\_delta', 'mean\_MFCC\_2nd\_coef', 'tqwt\_entropy\_log\_dec\_12', 'tqwt\_stdValue\_dec\_12', 'tqwt\_minValue\_dec\_13', 'tqwt\_stdValue\_dec\_11', 'tqwt\_maxValue\_dec\_11', 'tqwt\_minValue\_dec\_11', 'tqwt\_maxValue\_dec\_12', 'tqwt\_minValue\_dec\_12'] # Add all the specified features

tr\_x, tr\_y = tr\_psv[features], tr\_psv["class"] # Replace "target\_column" with your actual target column name

tst\_x, tst\_y = tst\_psv[features], tst\_psv["class"]

# 4. Normalize features scaler = StandardScaler()

tr\_x = scaler.fit\_transform(tr\_x) tst\_x = scaler.transform(tst\_x)

# 5. Hyperparameter tuning (example using GridSearchCV) xgb\_param\_grid = {

"n\_estimators": [100, 200, 300],

"max\_depth": [3, 5, 7],

# ... other XGBoost hyperparameters

}

et\_param\_grid = {

"n\_estimators": [100, 200, 300],

"max\_depth": [5, 8, 10],

# ... other ExtraTrees hyperparameters

}

xgb\_grid\_search = GridSearchCV(xgb.XGBClassifier(), xgb\_param\_grid, cv=5) xgb\_grid\_search.fit(tr\_x, tr\_y)

best\_xgb\_params = xgb\_grid\_search.best\_params\_

et\_grid\_search = GridSearchCV(ExtraTreesClassifier(), et\_param\_grid, cv=5) et\_grid\_search.fit(tr\_x, tr\_y)

best\_et\_params = et\_grid\_search.best\_params\_

# Create base models with best hyperparameters xgb\_model = xgb.XGBClassifier(\*\*best\_xgb\_params) et\_model = ExtraTreesClassifier(\*\*best\_et\_params)

# 6. Create ensemble model with voting

ensemble\_model = VotingClassifier(estimators=[("xgb", xgb\_model), ("et", et\_model)], voting="soft")

# 7. Train ensemble model ensemble\_model.fit(tr\_x, tr\_y)

# 8. Predict target class

predictions = ensemble\_model.predict(tst\_x)

# 9. Compute performance metrics

precision = precision\_score(tst\_y, predictions) recall = recall\_score(tst\_y, predictions)

f1 = f1\_score(tst\_y, predictions)

accuracy = accuracy\_score(tst\_y, predictions)

print("Accuracy:", accuracy) print("Classification Report:") print(classification\_report(tst\_y, predictions))

ENSEMBLE F-CORR

import pandas as pd import numpy as np

from sklearn.model\_selection import train\_test\_split from sklearn.ensemble import ExtraTreesClassifier from xgboost import XGBClassifier

from sklearn.ensemble import VotingClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

# Load the dataset from a CSV file

data = pd.read\_csv('/content/pd\_speech\_features.csv') # Define the features and the target variable

X = data[['gender', 'DFA', 'RPDE', 'meanPeriodPulses', 'stdDevPeriodPulses', 'locPctJitter', 'locAbsJitter', 'rapJitter', 'ppq5Jitter', 'ddpJitter', 'locShimmer', 'mean\_MFCC\_2nd\_coef', 'mean\_MFCC\_3rd\_coef', 'mean\_MFCC\_4th\_coef', 'mean\_MFCC\_5th\_coef', 'mean\_MFCC\_7th\_coef', 'mean\_MFCC\_8th\_coef', 'mean\_MFCC\_9th\_coef', 'mean\_MFCC\_11th\_coef', 'mean\_MFCC\_12th\_coef', 'mean\_4th\_delta', 'mean\_5th\_delta', 'mean\_6th\_delta', 'mean\_7th\_delta', 'mean\_10th\_delta', 'mean\_11th\_delta', 'mean\_12th\_delta', 'mean\_1st\_delta\_delta', 'mean\_2nd\_delta\_delta', 'mean\_4th\_delta\_delta', 'mean\_5th\_delta\_delta', 'mean\_6th\_delta\_delta', 'mean\_9th\_delta\_delta', 'mean\_10th\_delta\_delta', 'std\_Log\_energy', 'std\_MFCC\_0th\_coef', 'std\_MFCC\_1st\_coef', 'std\_MFCC\_2nd\_coef', 'std\_MFCC\_3rd\_coef', 'std\_MFCC\_4th\_coef', 'std\_MFCC\_5th\_coef', 'std\_MFCC\_6th\_coef', 'std\_MFCC\_7th\_coef', 'std\_MFCC\_8th\_coef', 'tqwt\_kurtosisValue\_dec\_13', 'tqwt\_kurtosisValue\_dec\_14', 'tqwt\_kurtosisValue\_dec\_15', 'tqwt\_kurtosisValue\_dec\_16', 'tqwt\_kurtosisValue\_dec\_17', 'tqwt\_kurtosisValue\_dec\_18', 'tqwt\_kurtosisValue\_dec\_19', 'tqwt\_kurtosisValue\_dec\_20', 'tqwt\_kurtosisValue\_dec\_21', 'tqwt\_kurtosisValue\_dec\_22', 'tqwt\_kurtosisValue\_dec\_23', 'tqwt\_kurtosisValue\_dec\_32', 'tqwt\_kurtosisValue\_dec\_33', 'tqwt\_kurtosisValue\_dec\_34', 'tqwt\_kurtosisValue\_dec\_35', 'tqwt\_kurtosisValue\_dec\_36']] # Include all the features here

y = data['class'] # Replace 'target\_variable' with the name of your target variable

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize the Extra Randomized Trees and XGBoost models model1 = ExtraTreesClassifier(n\_estimators=100, random\_state=0) model2 = XGBClassifier()

# Create an ensemble of models using VotingClassifier

ensemble\_model = VotingClassifier(estimators=[('et', model1), ('xgb', model2)], voting='soft')

# Train the ensemble model ensemble\_model.fit(X\_train, y\_train)

# Make predictions

y\_pred = ensemble\_model.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred) print("Accuracy:", accuracy)

# Calculate precision, recall, and F1-score precision = precision\_score(y\_test, y\_pred) recall = recall\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred)

print("Precision:", precision) print("Recall:", recall) print("F1-score:", f1)

// ENSEMBLE BOA

import pandas as pd import numpy as np

from sklearn.model\_selection import train\_test\_split, GridSearchCV from sklearn.ensemble import ExtraTreesClassifier

from xgboost import XGBClassifier

from sklearn.ensemble import VotingClassifier from sklearn.metrics import accuracy\_score from sklearn.impute import SimpleImputer

from sklearn.preprocessing import StandardScaler

from sklearn.feature\_selection import SelectFromModel

from sklearn.metrics import precision\_score, recall\_score, f1\_score

# Load the dataset from a CSV file

data = pd.read\_csv('/content/pd\_speech\_features.csv')

# Handle missing values

imputer = SimpleImputer(strategy='mean')

data\_imputed = pd.DataFrame(imputer.fit\_transform(data), columns=data.columns)

# Select the specified features

selected\_features = ['tqwt\_skewnessValue\_dec\_4', 'tqwt\_maxValue\_dec\_18', 'tqwt\_meanValue\_dec\_30', 'tqwt\_stdValue\_dec\_15', 'tqwt\_minValue\_dec\_11', 'tqwt\_minValue\_dec\_8', 'tqwt\_entropy\_log\_dec\_31', 'app\_entropy\_shannon\_7\_coef', 'ppq5Jitter', 'app\_LT\_entropy\_log\_6\_coef', 'tqwt\_stdValue\_dec\_10', 'GNE\_SNR\_TKEO', 'tqwt\_TKEO\_mean\_dec\_21', 'tqwt\_medianValue\_dec\_11', 'IMF\_SNR\_SEO', 'tqwt\_kurtosisValue\_dec\_12', 'mean\_9th\_delta\_delta', 'tqwt\_entropy\_log\_dec\_23', 'tqwt\_kurtosisValue\_dec\_31', 'tqwt\_medianValue\_dec\_18', 'tqwt\_skewnessValue\_dec\_33', 'mean\_0th\_delta', 'tqwt\_entropy\_log\_dec\_24', 'ddpJitter', 'Ed\_2\_coef', 'tqwt\_TKEO\_std\_dec\_36', 'tqwt\_meanValue\_dec\_32', 'std\_0th\_delta', 'tqwt\_entropy\_log\_dec\_6', 'f3', 'app\_entropy\_shannon\_5\_coef', 'IMF\_NSR\_entropy', 'tqwt\_maxValue\_dec\_10', 'std\_MFCC\_9th\_coef', 'mean\_12th\_delta', 'tqwt\_skewnessValue\_dec\_20', 'det\_LT\_entropy\_log\_5\_coef', 'tqwt\_TKEO\_mean\_dec\_6', 'tqwt\_entropy\_log\_dec\_5', 'tqwt\_medianValue\_dec\_17', 'tqwt\_medianValue\_dec\_2']

X = data\_imputed[selected\_features]

y = data['class'] # Replace 'target\_column' with your actual target column

# Drop rows with NaN values in the target variable nan\_rows = y[y.isnull()].index

X = X.drop(nan\_rows) y = y.drop(nan\_rows)

# Set a random seed for reproducibility np.random.seed(42)

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2) # Feature Scaling

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train) X\_test\_scaled = scaler.transform(X\_test)

# Feature Selection

clf = ExtraTreesClassifier() selector = SelectFromModel(clf)

X\_train\_selected = selector.fit\_transform(X\_train\_scaled, y\_train) X\_test\_selected = selector.transform(X\_test\_scaled)

# Hyperparameter Tuning for ExtraTreesClassifier param\_grid\_et = {

'n\_estimators': [100, 200, 300],

'max\_depth': [None, 5, 10],

'min\_samples\_split': [2, 5, 10]

}

grid\_search\_et = GridSearchCV(ExtraTreesClassifier(), param\_grid\_et, cv=5) grid\_search\_et.fit(X\_train\_selected, y\_train)

best\_model\_et = grid\_search\_et.best\_estimator\_

# Hyperparameter Tuning for XGBClassifier param\_grid\_xgb = {

'n\_estimators': [100, 200, 300],

'max\_depth': [3, 5, 7],

'learning\_rate': [0.1, 0.01, 0.001]

}

grid\_search\_xgb = GridSearchCV(XGBClassifier(), param\_grid\_xgb, cv=5) grid\_search\_xgb.fit(X\_train\_selected, y\_train)

best\_model\_xgb = grid\_search\_xgb.best\_estimator\_

# Create a Voting Classifier

voting\_clf = VotingClassifier(estimators=[('et', best\_model\_et), ('xgb', best\_model\_xgb)], voting='soft')

voting\_clf.fit(X\_train\_selected, y\_train)

# Make predictions and evaluate the model y\_pred = voting\_clf.predict(X\_test\_selected) accuracy = accuracy\_score(y\_test, y\_pred) precision = precision\_score(y\_test, y\_pred) recall = recall\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred)

print("Accuracy: {:.2f}".format(accuracy)) print("Precision:{:.2f}".format(precision)) print("Recall:{:.2f}".format(recall))

print("F1-score:{:.2f}".format(f1))

## REFERENCES

1. K Aditya Shastry, “An ensemble nearest neighbor boosting technique for prediction of Parkinson’s disease”, Healthcare Analytics, 2023, https:

//[www.sciencedirect.com/science/article/pii/S27724425](http://www.sciencedirect.com/science/article/pii/S27724425)

1. S. Yadav, M.K. Singh, S. Pal, “Artificial intelligence model for Parkinson disease detection using machine learning algorithms”, Biomed Mater Devices (2023), <http://dx.doi.org/10.1007/s44174-023-00068-x>
2. T. Velmurugan, J. Dhinakaran, “A novel ensemble stacking learning algorithm for Parkinson’s disease prediction”, Math. Probl. Eng. 2022 (2022) 9209656, <http://dx.doi.org/10.1155/2022/9209656>
3. J. Healthc., “Feature selection algorithm and deep learning classifier for Parkinson’s disease”, Eng. 2022 (2022) 1487212, [http://dx.doi.org/10. 1155/2022/1487212](http://dx.doi.org/10.%201155/2022/1487212)
4. Yang, F. Wang, L. Yang, et al., ”The physical significance of acoustic parameters and its clinical significance of dysarthria in Parkinson’s disease”, Sci. Rep. 10 (2020) 11776, <http://dx.doi.org/10.1038/> s41598-020-68754-0
5. Mei Jie, Desrosiers Christian, Frasnelli Johannes, “Machine learning for the diagnosis of Parkinson’s disease: A review of literature”, Front. Aging Neurosci. 13 (2021) 184, <http://dx.doi.org/10.3389/fnagi.2021.633752>
6. Simonson W, Hauser RA, Schapira AHV. “Role of the pharmacist in the effective management of wearing-off in Parkinson’s disease”, Ann Pharmacother. <https://journals.sagepub.com/doi/abs/10.1345/aph.1k348>
7. Dr. DeMaagd, “Parkinson’s disease and its management”, National Center of Biotechnology Information, 2015, https:/[/www.ncbi.nlm.nih.gov/pmc/](http://www.ncbi.nlm.nih.gov/pmc/) articles/PMC4517533/
8. A. Pramanik and A. Sarker, “Parkinson’s disease detection from voice and speech data using machine learning”, Proceedings of International Joint Conference on Advances in Computational Intelligence: IJCACI 2020, Springer Singapore, 2021, https://link.springer.com/chapter/ 10.1007/978-981-16-0586-4 36
9. L. NARANJO, C.J. PEREZ, J. MART ´ ´IN, Y. CAMPOS-ROCA, “A twostage

variable selection and classification approach for Parkinson’s disease detection by using voice recording replications”, 2017, Comput Methods Programs Biomed, <https://pubmed.ncbi.nlm.nih.gov/28325442/>

1. L. Ali, C. Chakraborty, Z. He, et al., “A novel sample and feature dependent ensemble approach for Parkinson’s disease detection”, Neural Comput. Appl. (2022), <http://dx.doi.org/10.1007/s00521-022-07046-2>
2. K. Taunk, S. De, S. Verma, A. Swetapadma, “A brief review of nearest neighbor algorithm for learning and classification”, in: 2019 International Conference on Intelligent Computing and Control Systems, ICCS, 2019, pp. 1255–1260, <http://dx.doi.org/10.1109/ICCS45141.2019.9065747>
3. W. Wang, J. Lee, F. Harrou, Y. Sun, “Early detection of Parkinson’s disease using deep learning and machine learning”, IEEE Access 8 (2020) 147635-147646, <http://dx.doi.org/10.1109/ACCESS.2020.3016062>
4. N. Radha, R.M. Sachin Madhavan, S. Sameera holy, ”Parkinson’s disease detection using machine learning techniques”, 30(2), 2021, [http:](http://dx.doi.org/10.1109/ICCS45141.2019.9065747%5b)

[//dx.doi.org/10.24205/03276716.2020.4055](http://dx.doi.org/10.1109/ICCS45141.2019.9065747%5b)

1. M.B. Makarious, H.L. Leonard, D. Vitale, et al., ”Multi-modality machine learning predicting Parkinson’s disease”, NPJ Parkinsons Dis. 8 (2022) 35, <http://dx.doi.org/10.1038/s41531-022-00288-w>

Parkinson disease prediction using Machine Learning

ORIGINALITY REPORT

33

%

SIMILARITY INDEX

25%

INTERNET SOURCES

25%

PUBLICATIONS

23%

STUDENT PAPERS

PRIMARY SOURCES

# rstudio-pubs-static.s3.amazonaws.com

1

3

Internet Source %

# 2 Submitted to Liverpool John Moores University

2%

Student Paper

# Submitted to University of North Texas

3

2

Student Paper %

# 4 K Aditya Shastry. "An ensemble nearest neighbor boosting technique for prediction of Parkinson’s disease", Healthcare Analytics, 2023

2%

Publication

# Submitted to University of Hertfordshire

5

1

Student Paper %

# dokumen.pub

6

1

Internet Source %

# 7 Abdelaziz Testas. "Distributed Machine Learning with PySpark", Springer Science and Business Media LLC, 2023

1%

Publication