MIT | Academy of Engineering
(An Autonomous Institute Affiliated to Savitribai Phule Pune University)

A Project Report on

Detection Of Parkinson's Disease Through Speech Analysis

Submitted by,

Ajinkya Kshatriya	(Exam Seat No. B234086)
Harshwardhan Tilekar	(Exam Seat No. B234006)
Piyush Bhondave	(Exam Seat No. B234087)
Sarthak Wakchaure	(Exam Seat No. B234102)

Guided by,

Mrs. Kavitha S

A Report submitted to MIT Academy of Engineering, Alandi(D), Pune, An Autonomous Institute Affiliated to Savitribai Phule Pune University in partial fulfillment of the requirements of

BACHELOR OF TECHNOLOGY in
Computer Engineering

School of Computer Engineering
MIT Academy of Engineering
(An Autonomous Institute Affiliated to Savitribai Phule Pune University)
Alandi (D), Pune – 412105

(2023–2024)

MIT | Academy of Engineering

(An Autonomous Institute Affiliated to Savitribai Phule Pune University)

CERTIFICATE

It is hereby certified that the work which is being presented in the BTECH Project Report entitled **"Detection Of Parkinson's Disease Through Speech Analysis"**, in partial fulfillment of the requirements for the award of the Bachelor of Technology in Computer Engineeringand submitted to the **School of Computer Engineering** of **MIT Academy of Engineering, Alandi(D), Pune, Affiliated to Savitribai Phule Pune University (SPPU), Pune**, is an authentic record of work carried out during Academic Year **2023–2024**, under the supervision of **Mrs. Kavitha S**, **School of Computer Engineering**

Ajinkya Kshatriya	**(Exam Seat No. B234086)**
Harshwardhan Tilekar	**(Exam Seat No. B234006)**
Piyush Bhondave	**(Exam Seat No. B234087)**
Sarthak Wakchaure	**(Exam Seat No. B234102)**

Mrs. Kavitha S	**Mr. Pranav Shriram**	**Dr. Rajeshwari Goudar**
Project Advisor	**Project Coordinator**	**Dean SCE**

Director/Dy. Director(AR)

External Examiner

DECLARATION

We the undersigned solemnly declare that the project report is based on our own work carried out during the course of our study under the supervision of **Mrs. Kavitha S**.

We assert the statements made and conclusions drawn are an outcome of our project work. We further certify that

1. The work contained in the report is original and has been done by us under the general supervision of our supervisor.
2. The work has not been submitted to any other Institution for any other degree/diploma/certificate in this Institute/University or any other Institute/University of India or abroad.
3. We have followed the guidelines provided by the Institute in writing the report.
4. Whenever we have used materials (data, theoretical analysis, and text) from other sources, we have given due credit to them in the text of the report and giving their details in the references.

Ajinkya Kshatriya	**(Exam Seat No. B234086)**
Harshwardhan Tilekar	**(Exam Seat No. B234006)**
Piyush Bhondave	**(Exam Seat No. B234087)**
Sarthak Wakchaure	**(Exam Seat No. B234102)**

Abstract

Parkinson's Disease (PD) patients' vocal alterations might be identified early on, allowing for management before physically incapacitating symptoms appear. In this work, static and dynamic speech characteristics that are relevant to PD identification are examined. The amount of articulation transitions and the trend of the fundamental frequency curve are considerably different between HC speakers and PD patients, according to a comparison of articulation transition features.The energy content in the transition from unvoiced to voiced segments (onset) and in the transition from voiced to unvoiced segments is used to calculate the dynamic speech characteristics (offset). This project proposes a method to build machine learning models using static features in terms of accuracy of PD detection under the two evaluation methods of splitting the dataset without samples overlap of one individual and 10-fold cross validation (CV).

Acknowledgment

We want to express our gratitude towards our respected project guide Mrs. Kavitha S for her constant encouragement and valuable guidance during the completion of this project work. We also want to express our gratitude towards respected School Dean Dr. Rajeshwari Goudar for her continuous encouragement. We would be failing in our duty if we do not thank all the other staff and faculty members for their experienced advice and evergreen co-operation

Ajinkya Kshatriya	**(Exam Seat No. B234086)**
Harshwardhan Tilekar	**(Exam Seat No. B234006)**
Piyush Bhondave	**(Exam Seat No. B234087)**
Sarthak Wakchaure	**(Exam Seat No. B234102)**

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Parkinson's disease is a neurodegenerative condition brought on by the loss of the neurotransmitter dopamine. PD is more prevalent. Gait and postural alterations in the aged population might increase the risk of falling and lead to mobility issues. It therefore affects daily activities and decreases the quality of life for patients and their families. Parkinson's illness mostly affects motor function. The signs of this movement disease include the inability to move freely, reduced and slow movement, increased muscular tonus, and shaking movement during rest. Lack of facial expression, poor coordination, and noticeable voice and speech changes are other traits. Since PD symptoms intensify as the disease advances, more sensitive diagnostic techniques are needed for PD diagnosis. An someone with, let's say, Parkinson's disease (PD) patients experience less stress, less intensity, and monotonous pitch and loudness (dysphonia). Given how easy and painless it is to gather speech data using mobile devices, the range of voice-related symptoms appears intriguing as a prospective screening approach. The early signs of Parkinson's disease are subtle, making them difficult to identify.

An intriguing research topic is the automatic detection of PD from speech since it has the potential to be used as a reliable instrument for non-invasive diagnosis. The capacity to detect PD subtly has the potential to dramatically advance medical care. Non-invasive procedures have a number of advantages, but their main advantage is that they enable diagnosis outside of a hospital environment, which reduces the inconvenience and cost of PD patients' physical travels for medical testing. Applications requiring on-the-spot screening and remote health monitoring are among the most popular choices for PD detection from speech as a result of this advantage.

Most of the time, Parkinson's disease has no identified aetiology. Pathological alterations in dopaminergic neurons and the resulting neurochemical dysfunction have been shown to be the most salient features of this illness. The bulk of the dopamine-producing neurons in the brainstem come together to produce a black substance called the substantia nigra. This anatomical region helps to produce regular bodily movement and has close links to other deep brain areas. Reduced range of motion and changes in voluntary motion are caused by a lack of dopamine production in the dopaminergic neurons of the substantia nigra. Parkinson's illness presently has no known cure. The illness progresses in different ways and at various rates. The signs of Parkinson's disease can be managed with a variety of medications.

1.1 Background

If we see the methods how Parkinson's disease was detected before Machine Learning is involved then we will get the idea of necessity of new approaches for detection of this disease. Parkinson's disease is neurodegenerative disease having progress over the time which have some unary percentage of affected people above the age sixty. Parkinson disease have mainly effect in older age group like people above the age fifty and similar. And if we see the growth rate of population and age wise growth then we can say that the crowd having danger of this disease is also increasing day by day. So, scope of this project is large enough that we need to watch it as a topic of concern.

Analyzing signs which are based on tests in clinic and the observations noted in medical are used for the diagnosis of Parkinson's Disease (PD), which includes the describing the variety of motor symptoms. But subjectivity is something with which traditional diagnos- tic approaches mostly suffers because they are dependent of the analyzation of movements which are mostly unclear to us and so as hard for classification, heading towards possi- bility of wrong classification. Most of the times early symptoms which does not include movements of PD are minor and so as we can say that there are strong chances that they are because of some other reasons or conditions. Concluding this we can say that taking decision based on these symptoms will lead us to wrong direction, making it challenging the detection of Parkinson earlier.

To find out solutions to these problems and difficulties and to make it possible to detect PD in early stage through updating procedure which had been used for detection and identification of Parkinson's disease, machine learning methods are to be used in this project to head towards diagnosis and differential diagnosis for Parkinson's Disease. With an investigation of source and type of data, aims heading towards use of ML methods and the related outcomes which mostly lead us to classification. A common method used from machine learning for detecting Parkinson's disease is classification where we classify the Parkinson's affected person and a healthy person based on voice signal from a concerned person, which is basically main aim of this project.

1.2 Project Idea

We intend to create and develop an application for finding out patient of Parkinson's disease through extracting the features of speech and its examination which will be referred for classifying Parkinson affected persons in order to achieve Parkinson disease affected and healthy person. The different characteristics taken from speech at a synchronization level of the pitch can be used to measure the quantity of abnormal functioning in the muscular structure of vocal tract caused by PD. By applying different kind of algorithms over the data set for classification purpose and comparing among them to build an efficient machine learning model in order to detect Parkinson's Disease

1.3 Motivation

Speech analysis is emerging as a promising method for detecting Parkinson's disease due to its accessibility and affordability. Unlike invasive procedures or inconclusive clinical obser- vations, speech analysis provides a convenient and cost-effective diagnostic tool. Vocal char- acteristics such as hoarseness, reduced loudness, imprecise articulation, and a monotonous pitch can serve as indicators of Parkinson's disease. Analyzing these vocal features, along with speech patterns characterized by abnormal pauses, decreased speech rate, and irreg- ularities in rhythm and intonation, can differentiate individuals with Parkinson's disease from healthy individuals. Advanced signal processing and machine learning techniques help capture and analyze these patterns accurately.

One significant advantage of speech analysis is its ability to provide objective measurements. Researchers can quantify various vocal and speech parameters, establishing quanti- tative metrics and algorithms to aid in diagnosing and monitoring Parkinson's disease. This approach reduces subjectivity and variability associated with traditional diagnostic meth- ods. Moreover, speech analysis enables remote monitoring of individuals with Parkinson's disease. Using smartphones or wearable devices, individuals can record speech samples at home and securely transmit them to healthcare professionals for analysis. This allows for continuous monitoring and timely intervention, enhancing patient convenience and care.

In summary, speech analysis offers accessible and non-invasive means of detecting Parkin- son's disease. By analyzing vocal characteristics and speech patterns, objective measure- ments can be obtained, leading to standardized and reproducible diagnostic approaches.

Additionally, remote monitoring through speech analysis facilitates timely intervention and improves patient outcomes.

1.4 Proposed Solution

The primary goal is to detect PD, which will be advantageous for patients who are suffering from Parkinson's disease and will lower the disease's prevalence. The architecture diagram outlines the process flow that is used to clean up the raw data and use it to forecast Parkinson's data. The collected raw data is first preprocessed into a format that can be understood. After that, we must divide the dataset into train and test data in order to train the data. The classification accuracy of this model is determined by evaluating the Parkinson's data using a machine learning algorithm that combines SVM, KNN, and Random forest.

Chapter 2

Literature Review

Serial No.	01
Title	Deep Learning-Based Parkinson's Disease Classification Using Vocal Feature Sets
Date	August 2019
Author	Hakan Gunduz (Gunduz, 2019)
Contribution of Author	In this study, CNN and SVM classifiers are used to classify PD.
Remarks	Choose the pertinent features that best capture the inherent characteristics of the speech (audio) data.

Serial No.	02
Title	On Detection Parkinson's Disease using Speech Signal of a phone call.
Date	August 2022
Author	Victor Monzon Baeza
Contribution of Author	The author has looked at a technique for utilizing a phone to diagnose Parkinson's symptoms. Signal processing can be used to save costs while enhancing patient comfort.
Remarks	Using the fact that vocal dysfunction may be among the initial PD symptoms, speech may be a good indication for identifying persons with Parkinson's.

Serial No.	03
Title	A Deep learning Based Method for Parkinson's Disease Detection Using Dynamic Features of Speech Dataset: Manual
Date	July 2019
Author	Changqin quan , kang ren, and zhiwei luo (Quan, Ren, & Luo, 2021)
Contribution of Author	In this paper, they used Traditional ML method, Deep Learning and articulation transition of HC speaker and PD patient.
Remarks	With classification accuracy of 80, the monologue feature of speech is best for identifying early PD patients from healthy controls.

Serial No.	04
Title	End-2-End Modeling of speech and gait from patients with parkinson's disease
Date	April 2021
Author	J. C. Vasquez-Correa1,2 T. Arias-Vergara1,2, 3 P. Klumpp1 P. A. Perez-Toro1,2 J. R.
Contribution of Author	They were able to find that without having microphone of high quality parkinson's disease can be detected with smartphone's microphone.
Remarks	The suggested model in this study is based on CNNs and uses Melspectrograms as input to analyse speech signals.

Serial No.	05
Title	Unobtrusive monitoring of speech impairments of parkinson's disease patients through mobile devices
Date	August 2018
Author	T. Arias-Vergara, J.C. Vasquez-Correa
Contribution of Author	In the course of casual discussions over the phone, the patients' speech is recorded.
Remarks	No need to check separately in hospitals.

Serial No.	06
Title	Towards an automatic monitoring of the neurological state of parkinson's patients from speech
Date	January 2021
Author	J.R. Orozco-Arroyave, J.C. Vasquez-Correa
Contribution of Author	The correlation values found with the Spanish data are often greater than those with the other languages. It is possible that this set of speakers' higher UPDRS levels and more variability contribute to the ease with which the neurological condition may be predicted.
Remarks	What if the UPDRS scores and variability of the speakers were higher?

Serial No.	07
Title	A comparative analysis of speech signal processing algorithms for Parkinson's disease classification
Date	January 2019
Author	Sakar, C. O., Serbes, G., Gunduz, A., Tunc, H. C., Nizam,H., Sakar, B. E., ... Apaydin(Sakar et al., 2019)
Contribution of Author	The author compared the various speech signal processing techniques for PD identification. In their research, a brand-new function known as the adjustable Q-factor wavelet transform was presented.
Remarks	Kernel SVM reported the best accuracy of 86 for all feature subsets. For the purpose of detecting PD, they employed a web application.

Serial No.	08
Title	The Detection of Parkinson’s Disease From Speech Using Voice Source Information
Date	June 2020
Author	N. P. Narendra , Bj¨orn Schuller , Fellow, IEEE, and Paavo Alku , Fellow
Contribution of Author	In this paper, author has done classification with SVM and MLP classifiers. PC-GITA database used to detect PD.
Remarks	SVM classifiers were created utilising baseline characteristics known to characterise articulation, phonation, and prosody in order to detect PD.

Serial No.	09
Title	Classification of Parkinson's disease Using Pitch Synchronous
Date	January 2018
Author	Sai Bharadwaj Appakaya and Ravi Sankar
Contribution of Author	They presented a unique approach for feature extraction and analysis in this work.
Remarks	The findings demonstrate that the PD-affected vocal tract may be followed as it changes using the pitch synchronous analysis approach.

Serial No.	10
Title	Effect ofacousticconditions on algorithms to detect parkinson's disease from speech
Date	November 2021
Author	J. C. Vdsquez- Correa, SerrOrozco- Arroyave. F. Vargas-Bonilla, E. Noth
Contribution of Author	In this study, the efficacy of five different techniques for detecting PD in speech was assessed.PD patients vs. HC speakers were assessed using a variety of acoustic settings, and the effects of these variables were also examined.
Remarks	Background noise most substantially affects the classification process. It must be given particular attention in order to regularly check the neurological condition. Using dynamic compression and codecs can improve the outcomes.

Chapter 3

Problem Definition and Scope

3.1 Problem statement

To design and develop an application for the detection of parkinson's disease using speech analysis.

3.2 Scope and Major Constraints

Scope of our project :

1. To validate a set of features appropriate for reviewing patient recordings with Parkinson's disease.
2. To perform preprocessing operations on speech.
3. To build an efficient machine learning model for the early detection of Parkinson's disease.
4. To evaluate the performance analysis with existing feature selection algorithms, sampling techniques, and classifiers, and compare the results.

3.3 Hardware and Software Requirements

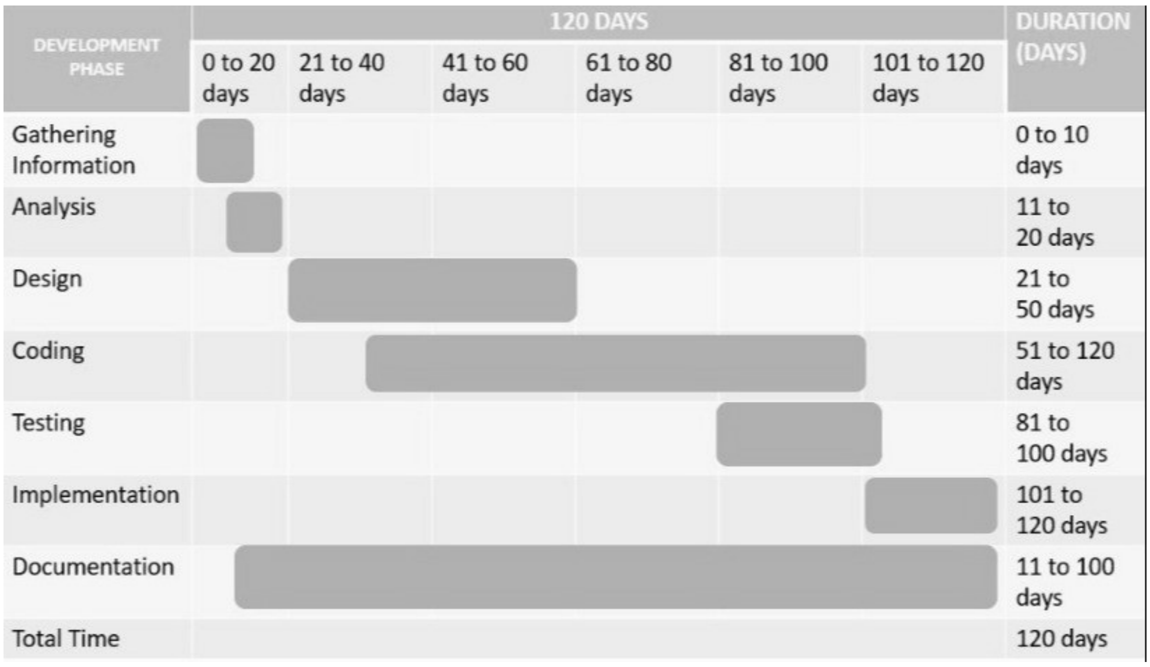
Software Requirements:

- Operating System: Windows, Linux
- Language: Python
- Libraries: Sklearn, Numpy, Pandas
- IDE: Google Colab
- Algorithm: SVM, KNN, Random Forest

Hardware Requirements:

- Device: PC or Laptop
- Equipment: Microphone

3.4 Action Plan



DEVELOPMENT PHASE	120 DAYS						DURATION (DAYS)
	0 to 20 days	21 to 40 days	41 to 60 days	61 to 80 days	81 to 100 days	101 to 120 days	
Gathering Information	▨						0 to 10 days
Analysis	▨						11 to 20 days
Design		▨	▨				21 to 50 days
Coding		▨	▨	▨	▨		51 to 120 days
Testing					▨	▨	81 to 100 days
Implementation						▨	101 to 120 days
Documentation	▨	▨	▨	▨	▨	▨	11 to 100 days
Total Time							120 days

Figure 3.1: Action Plan

Chapter 4

System Requirement Specification

4.1 Overall Description

The prerequisites are explained in this chapter. outlines the software and hardware specifications needed for the program to operate correctly. The assessment of the dissertation and its functional and non-functional requirements are provided in the Software Requirements Specification (SRS), which is detailed in depth.

An SRS document outlines each data, functional, and behavioral requirement for software that is under development or production. The SRS is a crucial document that forms the basis of the software development process. It gives a full description of how the behavior of the system will change. Along with a list of the system requirements, it also offers a synopsis of its main features. The aforementioned activities are a part of the systems engineering and software engineering process known as requirements analysis, which identifies the requirements that must be fulfilled for a new or modified product while taking into account the potentially conflicting requirements of various stakeholders, such as beneficiaries or users. The outcomes of the requirements analysis determine the success of a development project. Requirements should be recorded, requirement tested, tied to known business opportunities or requirements and stated in sufficient depth to support system design.

The SRS serves as a guide for finishing the project. The SRS is sometimes referred to as a "mother" document since it serves as the foundation for all other project management papers that come after it, including design specifications, statements of work, and software architecture. The SRS only includes functional and non-functional criteria, which is significant to notice.

4.2 Modeling

An engineering depiction of a proposed construction is known as a design. The most critical part of the development process is when requirements are represented as software. The best method for faithfully converting a customer's request into a final software solution is design. The design of a new structure must be backed by software, be based on the user's needs, and be thoroughly analyzed against the existing system. The system design is not yet complete. The end product of the design process is a representation or model that includes details on the architecture, interfaces, data formats, and other system-related components. It is necessary to make changes to the logical layout of the system, which was generated through system analysis, before creating the physical design.

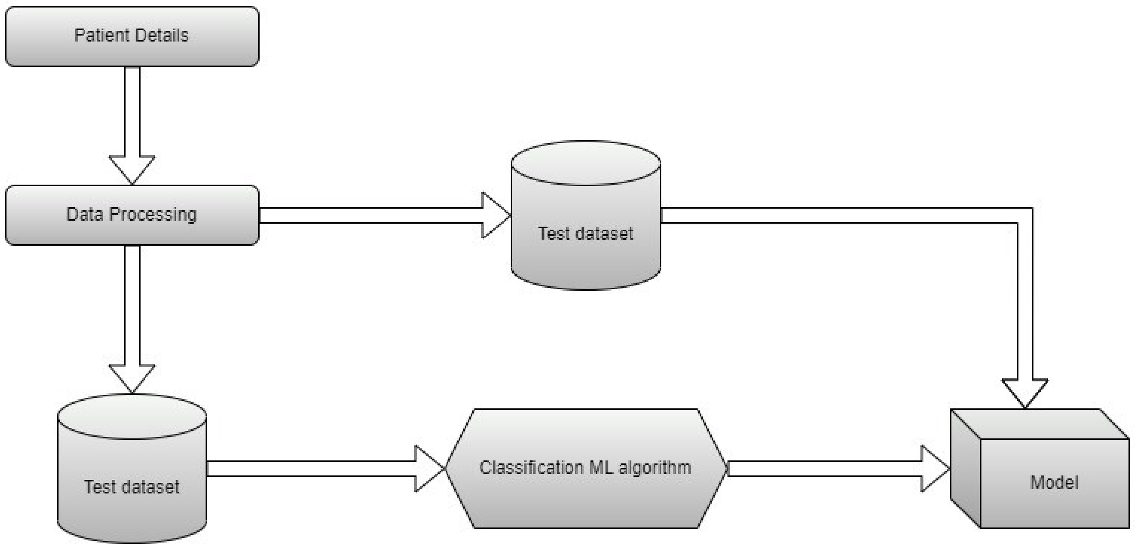


Figure 4.1: Block Diagram

4.2.1 Use Case Diagram

An interaction between external entities and the system under study that is goal-oriented is defined by a use case. The players in the system are external entities that communicate with it. A collection of use cases that can be graphically labeled describes the whole capabilities of the system in great detail.

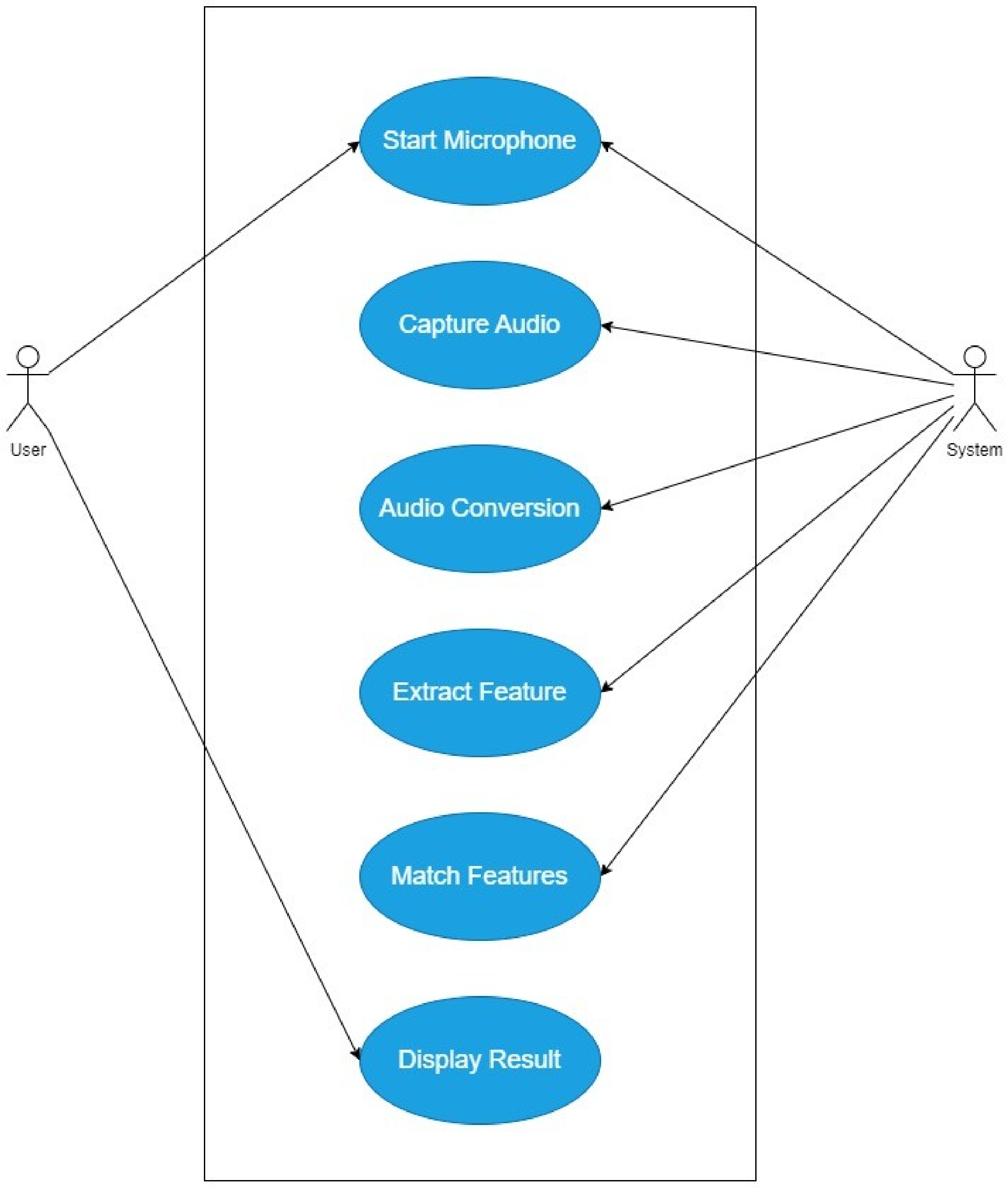


Figure 4.2: Use Case Diagram

4.2.2 Activity Diagram

An activity diagram displays the order in which a complicated process's various steps must occur. The name of the operation is presented in a round box that represents the activity. The transition signaled by completion is indicated by the designed arrow added to the end of the activity symbol.

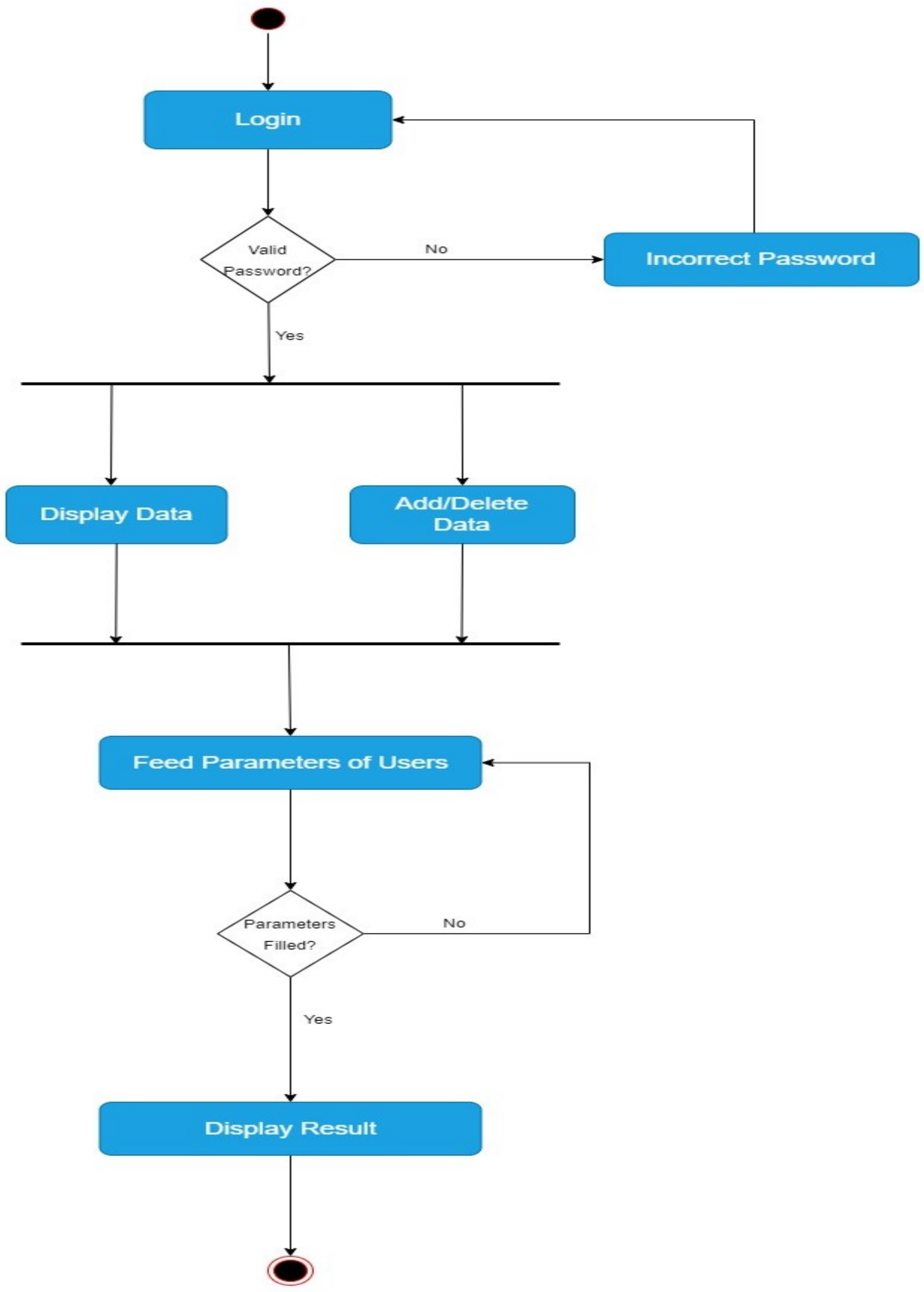


Figure 4.3: Activity Diagram

4.2.3 Sequence Diagram

A sequence diagram illustrates how and in what order a collection of items interacts, a sequence diagram is a form of interaction diagram. Software designers and business experts use these diagrams to clarify the specifications for a new system or to record an existing procedure. The terms "sequence diagrams" and "event scenarios" are frequently used interchangeably.

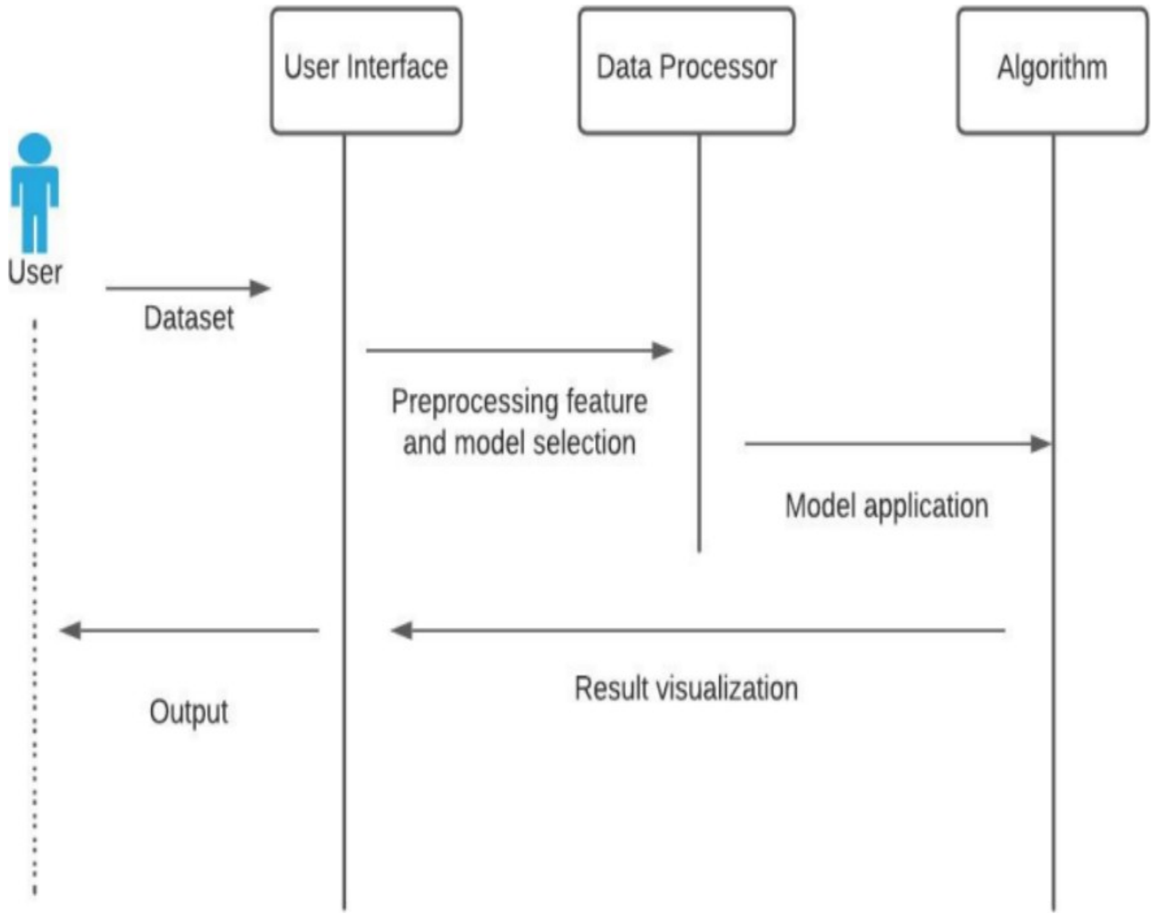


Figure 4.4: Activity Diagram

Chapter 5

Methodology

5.1 Overall Description

The prerequisites are explained in this chapter. outlines the hardware and software specifications needed for the application to operate correctly. A detailed explanation of the Software Requirements Specification(SRS), which comprises a summary of a dissertation and its functional and non-functional requirements, is provided.

All information, functional requirements, and behavioral requirements for software that is in production or development are described in an SRS document. A key document that serves as the foundation for the software development process is the SRS. It provides a thorough explanation of how the system's behavior will be evolved. It includes a description of its key features in addition to a list of the system requirements. These activities are a part of requirements analysis in software engineering and systems engineering, which identifies the requirements that must be fulfilled for a new or improved product while taking into consideration the potential compatibility of a subject with the other stakeholders, including such recipients or users. The success of a development project depends on the findings of a requirements analysis. The demand must be well-defined, include adequate details for system design, be measurable, testable, and linked to known business opportunities or needs.

5.2 System Architecture

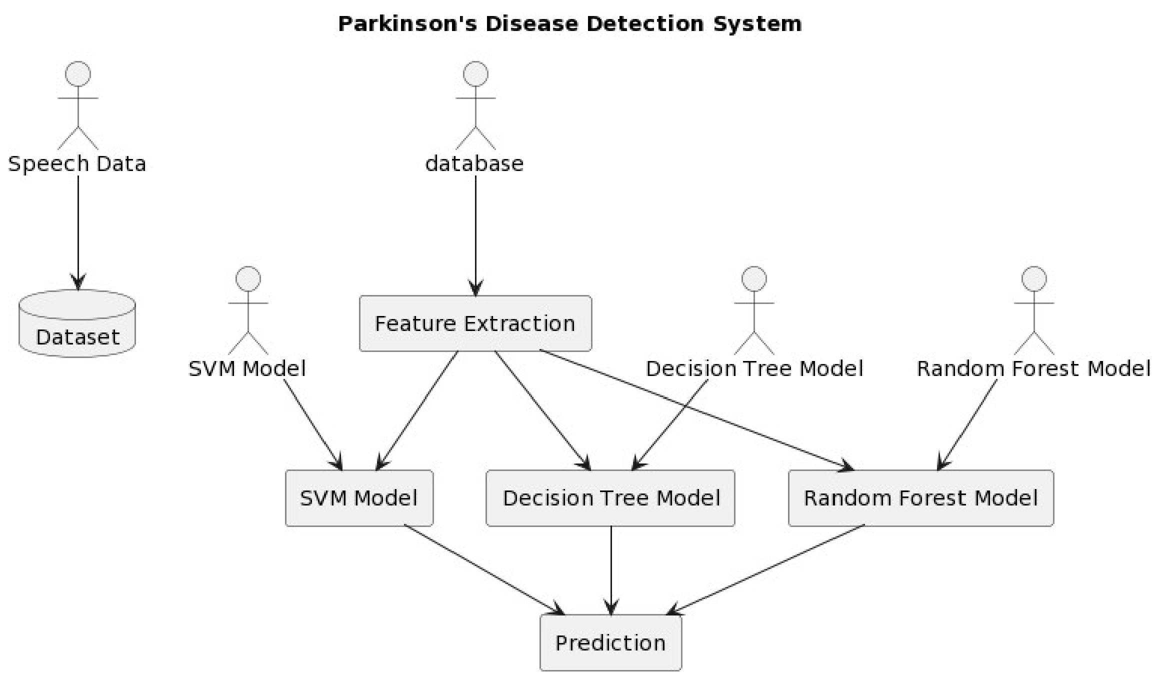


Figure 5.1: System Architecture

The diagram illustrates the architecture of a Parkinson's Disease Detection System. It consists of the following components: Dataset: This component represents the collection of speech data that serves as input for the system. The dataset contains samples of speech recordings from individuals, both with and without Parkinson's disease.

Feature Extraction: This component is responsible for extracting relevant features from the speech data. It processes the input speech recordings and derives a set of meaningful features that capture important characteristics related to Parkinson's disease.

SVM Model: This component represents the Support Vector Machine (SVM) model used for classification. It receives the extracted features from the Feature Extraction component and employs SVM algorithms to train a model that can distinguish between individuals with Parkinson's disease and those without. Decision Tree Model: This component represents the Decision Tree model used for classification. It takes the extracted features as input and constructs a decision tree based on the data. The decision tree splits the data based on feature values and creates a set of rules for classifying individuals with Parkinson's disease.

Random Forest Model: This component represents the Random Forest model used for classification. It utilizes an ensemble of decision trees to classify the input data. Each decision tree is trained on a subset of the features and samples, resulting in a diverse set of models. The predictions from multiple decision trees are combined to make the final classification. Prediction: This component represents the process of making predictions or classifications using the trained models. It takes the input speech data and passes it through the SVM Model, Decision Tree Model, and Random Forest Model to obtain predictions on whether an individual has Parkinson's disease or not.

The diagram highlights the flow of data and interactions between the components, indicating that the Dataset is processed by the Feature Extraction component, and the extracted features are used by the SVM Model, Decision Tree Model, and Random Forest Model for making predictions.

5.3 Mathematical Modeling

Support Vector Machine (SVM): The Support Vector Machine (SVM) algorithm is a powerful classification method that aims to find an optimal hyper-plane that separates the data points into different classes. The SVM model in the provided code uses a linear kernel and implements the soft-margin formulation. The SVM model seeks to find the weight vector w and bias term b that define the hyperplane equation: wx - b = 0, where x represents the input features. Given a training set of input features X and corresponding labels y, the goal is to find the optimal values of w and b. The SVM algorithm formulates the problem as an optimization task, aiming to maximize the margin between the hyperplane and the nearest data points of different classes. The optimization objective is subject to the constraint: y(wx - b) 1 for all training examples, where y represents the class label (-1 or 1) for each training example x. In the code, the gradient descent method is used to iteratively update the weight vector w and bias term b, based on the calculated gradients and the learning rate.

Decision Tree: A Decision Tree is a versatile machine-learning algorithm that can be used for both classification and regression tasks. It recursively partitions the feature space based on the values of input features, allowing for intuitive and interpretable decision-making. The decision tree algorithm builds a tree-like model where each internal node represents a decision based on a specific feature and threshold value. The splitting process involves finding the best feature and threshold that maximizes the information gain or Gini impurity reduction. At each internal node, a decision is made based on the condition: if the value of the selected feature is less than or equal to the threshold, go left; otherwise, go right. The process is repeated recursively until it reaches the leaf nodes, which contain the final predictions or class labels. The code implements a basic decision tree model that uses the information gain criterion for splitting and includes parameters such as the maximum depth and minimum number of samples required to split a node.

Random Forest: Random Forest is an ensemble learning algorithm that combines multiple decision trees to make predictions. Each decision tree is trained on a random subset of the data and features, providing robust and accurate predictions.

Random Forest builds an ensemble of decision trees by using the technique of bootstrap aggregating (or bagging) to create different subsets of the training data. Additionally, during the training of each decision tree, a random subset of features is selected, reducing the correlation between individual trees. During prediction, each decision tree in the forest independently provides a prediction, and the final prediction is determined by aggregating the predictions, typically using majority voting for classification tasks. Random Forest helps improve the overall performance and generalization of the model by reducing overfitting and increasing the stability of the predictions. The code implements a Random Forest model that creates and trains multiple decision trees using the provided parameters such as the number of trees, maximum depth, minimum samples required to split a node, and the number of features to consider for each split. In summary, the provided code includes implementations of the SVM, Decision Tree, and Random Forest algorithms for Parkinson's disease detection using speech data. These algorithms are widely used in machine learning for classification tasks and offer different approaches to modeling and prediction.

5.4 Approach/Algorithms

5.4.1 Decision tree

A decision tree is a popular machine-learning algorithm used for both classification and regression tasks. It is a flowchart-like structure that breaks down a dataset into smaller subsets based on different features, ultimately leading to a prediction or decision at the tree's leaves.

Here are some key points about decision trees:

Structure: A decision tree consists of nodes and branches. The root node represents the entire dataset, and subsequent nodes split the data based on specific features. Leaf nodes represent the final decision or prediction.

Splitting Criteria: Decision trees use different criteria to determine how to split the data at each node. Common splitting criteria include Gini impurity and entropy for classification tasks and mean squared error or variance reduction for regression tasks.

Feature Selection: Decision trees determine the best feature to split the data by evaluating the importance or relevance of each feature. This can be measured using metrics like information gain, gain ratio, or feature importance based on the chosen criterion.

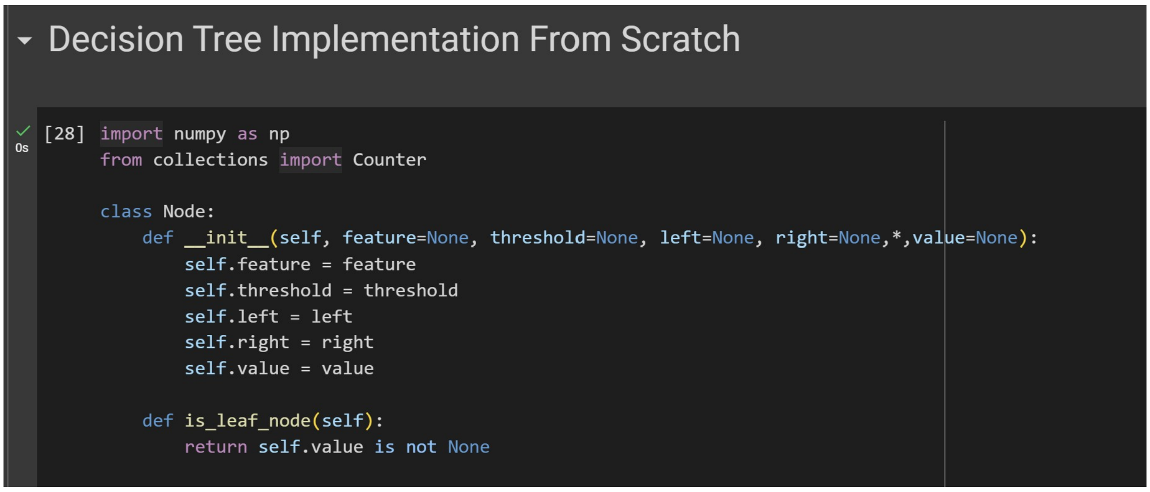
Handling Categorical and Numerical Data: Decision trees can handle both categorical and numerical features. For categorical features, the tree can simply split the data based on different categories. For numerical features, the tree selects a threshold value to split the data into two subsets.

Pruning: Decision trees can be prone to overfitting, where the model becomes too complex and performs poorly on unseen data. Pruning techniques, such as pre-pruning (stopping tree growth early) or post-pruning (removing unnecessary branches), help prevent overfitting and improve generalization. Ensemble Methods:

Decision trees are often combined in ensemble methods such as Random Forests or Gradient Boosting, which create multiple trees and aggregate their predictions for improved accuracy and robustness.

Interpretable and Explainable: Decision trees provide a transparent and interpretable representation of the decision-making process. The paths from the root to a leaf node can be easily understood and explainable, making decision trees valuable in domains where interpretability is crucial.

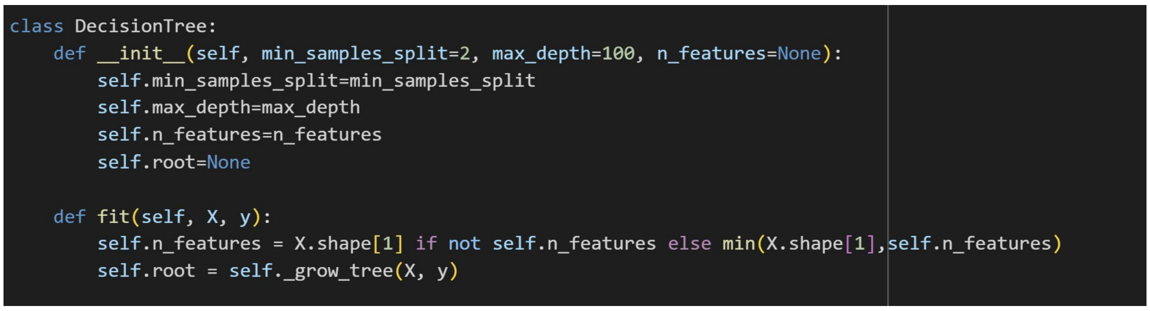
Overall, decision trees offer a versatile and interpretable approach to machine learning, with applications ranging from medical diagnosis to customer segmentation and fraud detection.



```
import numpy as np
from collections import Counter

class Node:
    def __init__(self, feature=None, threshold=None, left=None, right=None,*,value=None):
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value

    def is_leaf_node(self):
        return self.value is not None
```

Figure 5.2: Decision Tree Implementation

The code imports the numpy library for numerical operations and the Counter class from the collections module for counting occurrences of values. The ”Node” class is defined, which represents a node in the decision tree. It has several attributes: ”feature”: Represents the feature that the node splits on. ”threshold”: represents the threshold value for the feature. ”left” and ”right”: Represent the left and right child nodes of the current node. ”value”: Represents the predicted value or class at the leaf node. The ”init” method is a constructor that initializes the attributes of the ”Node” class. The feature, threshold, left, right, and value parameters are set based on the provided arguments. The ”is leaf node” method checks whether the current node is a leaf node. It returns True if the node has a value (i.e., it is a leaf node) and False otherwise.



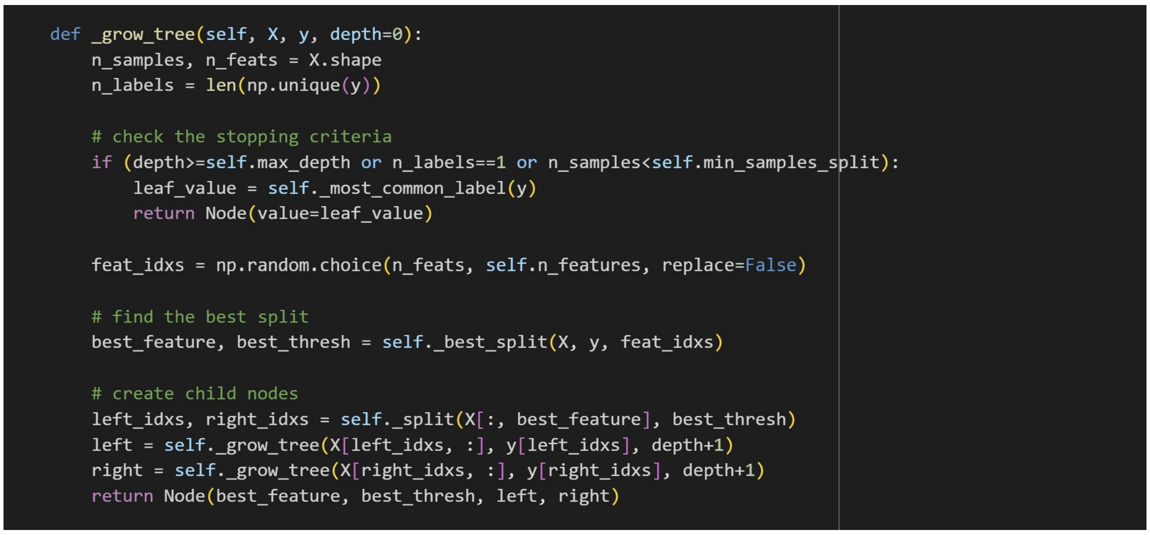
```
class DecisionTree:
    def __init__(self, min_samples_split=2, max_depth=100, n_features=None):
        self.min_samples_split=min_samples_split
        self.max_depth=max_depth
        self.n_features=n_features
        self.root=None

    def fit(self, X, y):
        self.n_features = X.shape[1] if not self.n_features else min(X.shape[1],self.n_features)
        self.root = self._grow_tree(X, y)
```

Figure 5.3: Decision Tree

The init method is the constructor of the DecisionTree class. It initializes the parameters for the decision tree algorithm, including min samples split, max depth, and n features. These parameters control the minimum number of samples required to split a node, the maximum depth of the tree, and the maximum number of features to consider when looking for the best split, respectively. The root attribute is set to None initially.

The fit method is used to train the decision tree model. It takes two arguments, X and y, representing the input features and target values, respectively. Inside the method, the number of features is determined based on the shape of X unless a specific number is provided. Then, the grow tree method is called to build the decision tree.

The grow tree method is a private method that recursively builds the decision tree. It takes the input features X and target values y as arguments. This method is responsible for splitting the data at each node based on the selected features and creating child nodes until a stopping criterion is met. The specific implementation of the grow tree is not provided in the code snippet.



```python
def _grow_tree(self, X, y, depth=0):
    n_samples, n_feats = X.shape
    n_labels = len(np.unique(y))

    # check the stopping criteria
    if (depth>=self.max_depth or n_labels==1 or n_samples<self.min_samples_split):
        leaf_value = self._most_common_label(y)
        return Node(value=leaf_value)

    feat_idxs = np.random.choice(n_feats, self.n_features, replace=False)

    # find the best split
    best_feature, best_thresh = self._best_split(X, y, feat_idxs)

    # create child nodes
    left_idxs, right_idxs = self._split(X[:, best_feature], best_thresh)
    left = self._grow_tree(X[left_idxs, :], y[left_idxs], depth+1)
    right = self._grow_tree(X[right_idxs, :], y[right_idxs], depth+1)
    return Node(best_feature, best_thresh, left, right)
```

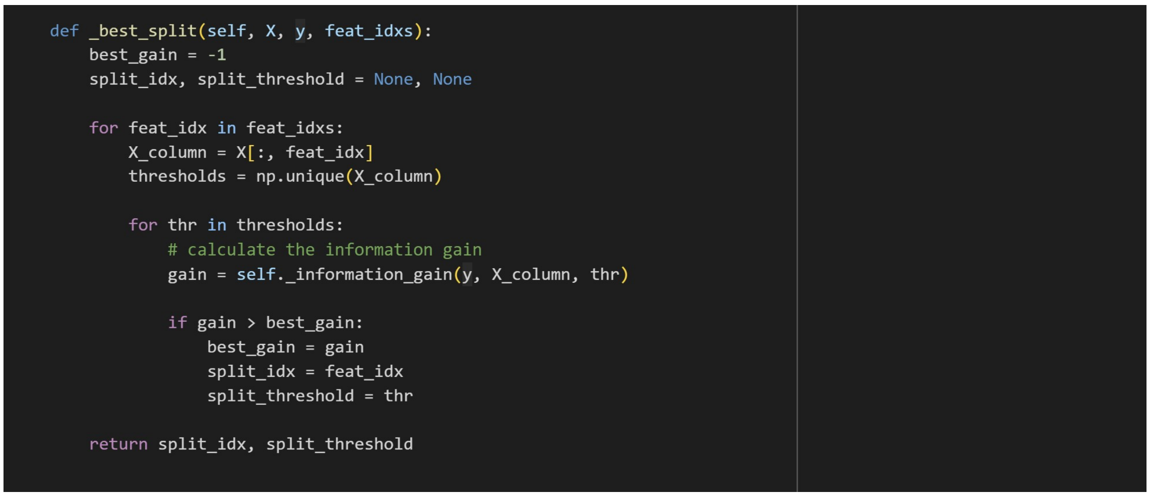
Figure 5.4: Grow Tree

The function takes as input the feature matrix X, the target vector y, and the current depth of the tree. It first calculates the number of samples (n samples), number of features (n feats), and number of unique labels (n labels) in the target vector.

Next, it checks the stopping criteria to determine if a leaf node should be created. The stopping criteria include reaching the maximum allowed depth (self.max depth), having only one unique label (n labels == 1), or having several samples below the minimum required for further splitting (n samples ¡ self.min samples split). If any of these conditions are met, a leaf node is created with the most common label in the target vector, and it is returned.

If the stopping criteria are not met, the code proceeds to randomly select a subset of features (self.n features) from the available features (feat idxs). The code then finds the best feature and threshold to split the data based on the selected subset of features. This is done by calling the internal function best split(), which determines the feature and threshold that maximize the information gain or minimize the impurity.

After finding the best split, the data is divided into left and right indices based on the selected feature and threshold. Recursively, the grow tree() function is called on the left and right subsets of the data to create child nodes. The depth is incremented by 1 in each recursive call. Finally, a node is created with the best feature, threshold, and the left and right child nodes, and it is returned.



```
def _best_split(self, X, y, feat_idxs):
    best_gain = -1
    split_idx, split_threshold = None, None

    for feat_idx in feat_idxs:
        X_column = X[:, feat_idx]
        thresholds = np.unique(X_column)

        for thr in thresholds:
            # calculate the information gain
            gain = self._information_gain(y, X_column, thr)

            if gain > best_gain:
                best_gain = gain
                split_idx = feat_idx
                split_threshold = thr

    return split_idx, split_threshold
```

Figure 5.5: Best Split

The best split() method takes the following parameters:

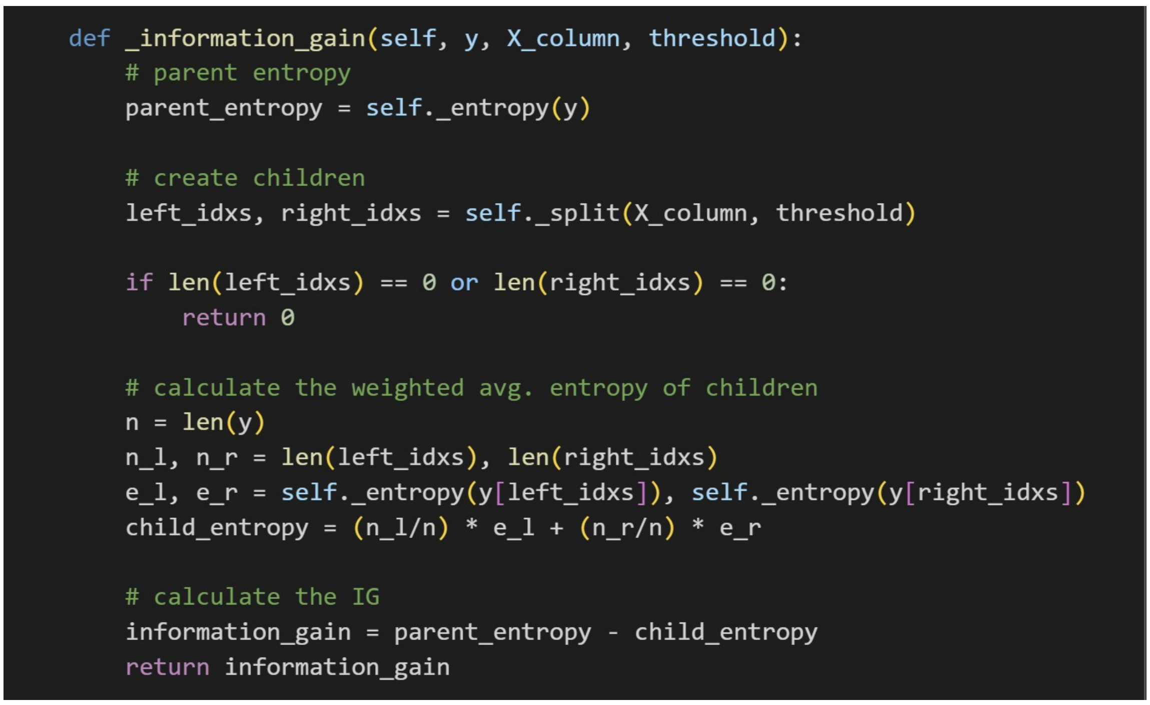
X: The feature matrix of the dataset. y: The target variable or labels corresponding to the features. feat idxs: A list of feature indices to consider for splitting. The code initializes variables' best gain, split idx, and split threshold to track the best information gain, the index of the feature for the split, and the threshold value for the split, respectively.

The method then iterates over each feature index in feat idxs. For each feature, it retrieves the corresponding column from the feature matrix X and finds unique values or thresholds within that column.

Next, it enters a nested loop, iterating over each threshold value. For each threshold, it calculates the information gain using a helper function information gain(), which is not shown in the given code snippet. The information gain measures the reduction in entropy or impurity after the split.

If the calculated gain is greater than the current best gain, it updates the best gain, split idx, and splits the threshold with the new values.

Finally, after examining all features and their thresholds, the method returns the split idx and split threshold representing the best-split point found in the dataset. The method takes three inputs: the target variable (y), a specific feature column (X column), and a threshold value.



```
def _information_gain(self, y, X_column, threshold):
    # parent entropy
    parent_entropy = self._entropy(y)

    # create children
    left_idxs, right_idxs = self._split(X_column, threshold)

    if len(left_idxs) == 0 or len(right_idxs) == 0:
        return 0

    # calculate the weighted avg. entropy of children
    n = len(y)
    n_l, n_r = len(left_idxs), len(right_idxs)
    e_l, e_r = self._entropy(y[left_idxs]), self._entropy(y[right_idxs])
    child_entropy = (n_l/n) * e_l + (n_r/n) * e_r

    # calculate the IG
    information_gain = parent_entropy - child_entropy
    return information_gain
```

Figure 5.6: Information Gain

The parent entropy variable is calculated by calling another method entropy(y). This method calculates the entropy of the target variable, which is a measure of the impurity or disorder in the dataset.

The code then proceeds to split the data into two subsets: left idxs and right idxs. These subsets are created based on whether the values in X column are less than or equal to the threshold or greater than the threshold, respectively. If either the left or right subset is empty (i.e., no data points fall into one of the subsets), the information gain is set to 0, as there would be no gain in splitting in such cases.

Next, the weighted average entropy of the children (left and right subsets) is calculated. The weights are determined by the proportion of data points in each subset relative to the total number of data points.

Finally, the information gain is computed as the difference between the parent entropy and the child entropy. Information gain measures the reduction in entropy achieved by splitting the data based on the given feature and threshold. A higher information gain indicates a more informative feature for the split. split(self, X column, split thresh): This function takes an input column X column from the dataset and a split threshold value split thresh. It splits the data into two subsets based on

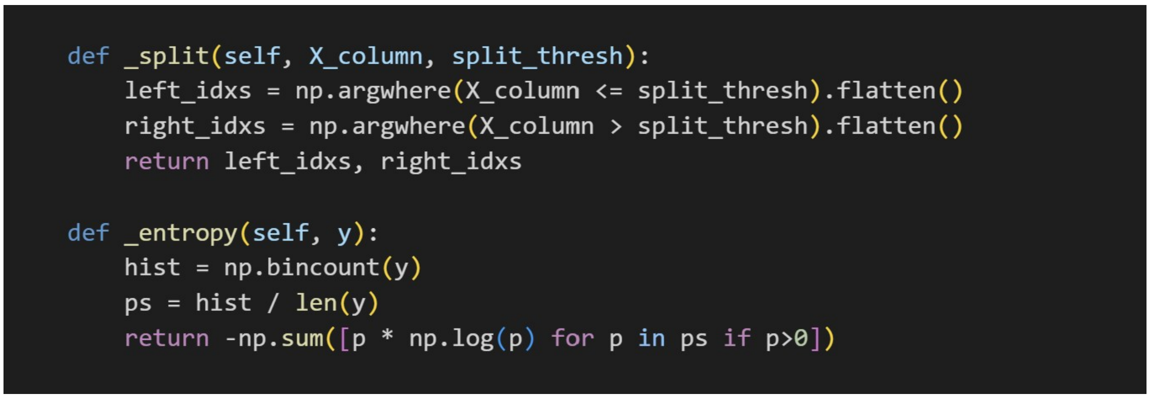
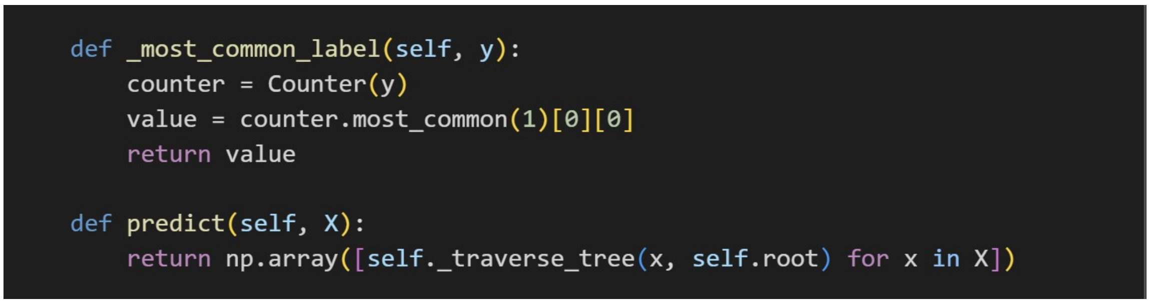

```python
def _split(self, X_column, split_thresh):
    left_idxs = np.argwhere(X_column <= split_thresh).flatten()
    right_idxs = np.argwhere(X_column > split_thresh).flatten()
    return left_idxs, right_idxs

def _entropy(self, y):
    hist = np.bincount(y)
    ps = hist / len(y)
    return -np.sum([p * np.log(p) for p in ps if p>0])
```

Figure 5.7: Split

the split threshold: values less than or equal to the threshold are assigned to the left subset, while values greater than the threshold are assigned to the right subset. The function uses the NumPy function np.argwhere() to find the indices of the elements satisfying the split condition, and then flattens the resulting array using flatten(). Finally, it returns the indices of the left and right subsets as left idxs and right idxs, respectively.

entropy(self, y): This function calculates the entropy of the target variable y. Entropy is a measure of the impurity or disorder in a set of data. The function first computes the histogram of y using np.bincount() to count the occurrences of each unique value in y. It then calculates the probabilities ps by dividing each count by the total number of elements in y. Finally, it calculates the entropy by summing the negative of each probability multiplied by the logarithm of the probability, using a list comprehension to exclude probabilities of zero. The negative sign is applied to ensure a positive entropy value. The resulting entropy value is returned.



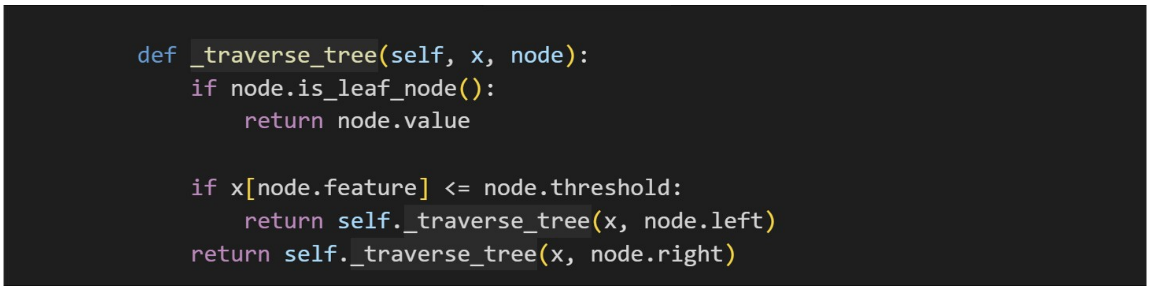
```
def _most_common_label(self, y):
    counter = Counter(y)
    value = counter.most_common(1)[0][0]
    return value

def predict(self, X):
    return np.array([self._traverse_tree(x, self.root) for x in X])
```

Figure 5.8: Most Common Label

most common label function:

This function takes in a list of labels (y) and determines the most common label in the list. It uses the Counter class from the collections module to count the occurrences of each label. The most common(1) method returns a list of tuples containing the most common label and its count. The function then retrieves the label from the first tuple ([0][0]) and returns it. predict method:

This method takes in a 2D array of input features (X) and predicts the corresponding labels using the decision tree. It uses a list comprehension to iterate over each input instance (x) in X and calls the traverse tree function with the instance and the root node of the decision tree. The predicted labels are collected in a NumPy array and returned as the final prediction.



```python
def _traverse_tree(self, x, node):
    if node.is_leaf_node():
        return node.value

    if x[node.feature] <= node.threshold:
        return self._traverse_tree(x, node.left)
    return self._traverse_tree(x, node.right)
```

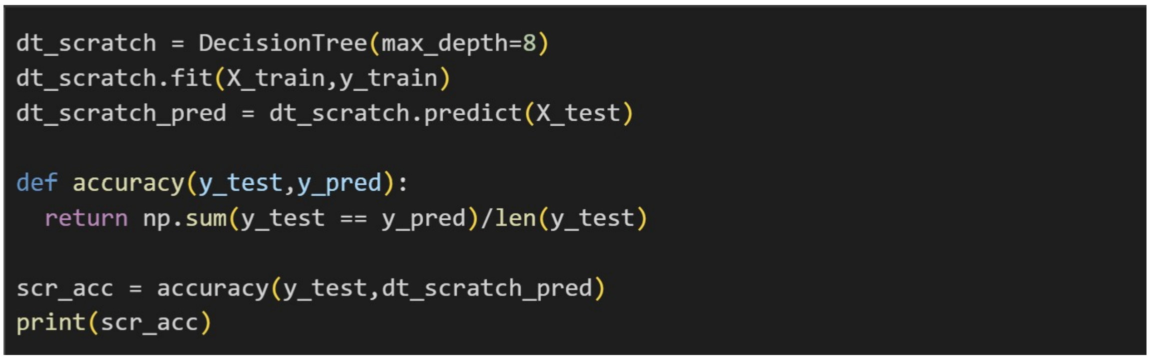
Figure 5.9: Traverse Tree

The function takes two parameters: x, which represents the input to be classified, and node, which represents the current node being evaluated in the decision tree.

The first condition checks if the current node is a leaf node, which means it represents a final decision or prediction. If it is a leaf node, the function returns the value associated with that node. If the current node is not a leaf node, the code proceeds to evaluate the input x based on the feature and threshold values associated with the current node.

If the value of the feature in x is less than or equal to the threshold value of the current node, the code recursively calls the traverse tree with x and the left child node of the current node. If the value of the feature in x is greater than the threshold value of the current node, the code recursively calls the traverse tree with x and the right child node of the current node.

The recursive process continues until a leaf node is reached, and the corresponding value is returned as the final prediction for the input x.



```
dt_scratch = DecisionTree(max_depth=8)
dt_scratch.fit(X_train,y_train)
dt_scratch_pred = dt_scratch.predict(X_test)

def accuracy(y_test,y_pred):
  return np.sum(y_test == y_pred)/len(y_test)

scr_acc = accuracy(y_test,dt_scratch_pred)
print(scr_acc)
```

Figure 5.10: Calling All Functions

dt scratch = DecisionTree(max depth=8): This line creates an instance of a decision tree classifier called dt scratch with a maximum depth of 8. The max depth parameter determines the maximum number of levels in the decision tree.

dt scratch.fit(X train, y train): This line trains the decision tree classifier using the training data X train and the corresponding labels y train. It learns to make predictions based on the input features X train and their associated target values y train.

dt scratch pred = dt scratch.predict(X test): This line uses the trained decision tree classifier (dt scratch) to predict the labels for the test data X test. The predicted labels are stored in the dt scratch pred variable.

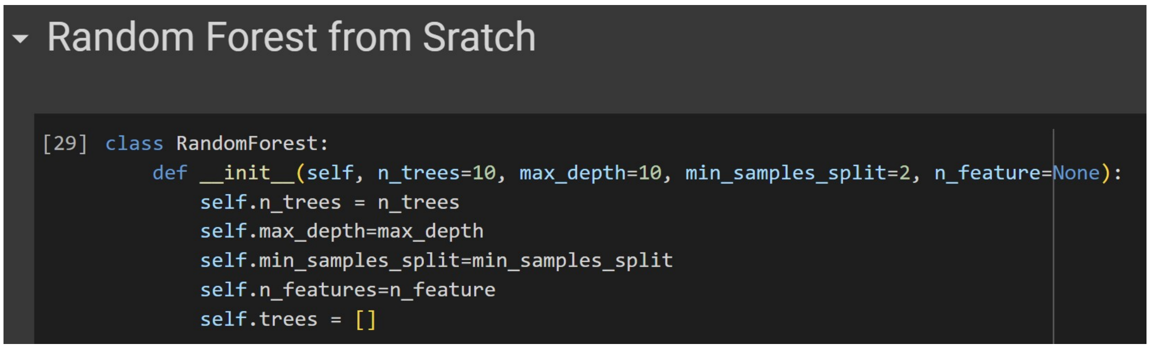
def accuracy(y test, y pred): This line defines a function called accuracy that takes two arguments: y test (the true labels) and y pred (the predicted labels).

return np.sum(y test == y pred) / len(y test): This line calculates the accuracy of the predictions by comparing each element of the true labels y test with the corresponding element in the predicted labels y pred. The function returns the ratio of the number of correct predictions to the total number of predictions.

scr acc = accuracy(y test, dt scratch pred): This line calls the accuracy function with the true labels y test and the predicted labels dt scratch pred, and assigns the resulting accuracy score to the variable scr ACC.

print(scr acc): This line prints the accuracy score (scr acc) on the console, which represents the proportion of correctly predicted labels in the test data.

5.4.2 Random Forest


Random Forest from Sratch

```
[29] class RandomForest:
         def __init__(self, n_trees=10, max_depth=10, min_samples_split=2, n_feature=None):
             self.n_trees = n_trees
             self.max_depth=max_depth
             self.min_samples_split=min_samples_split
             self.n_features=n_feature
             self.trees = []
```

Figure 5.11: Random Forest Implementation

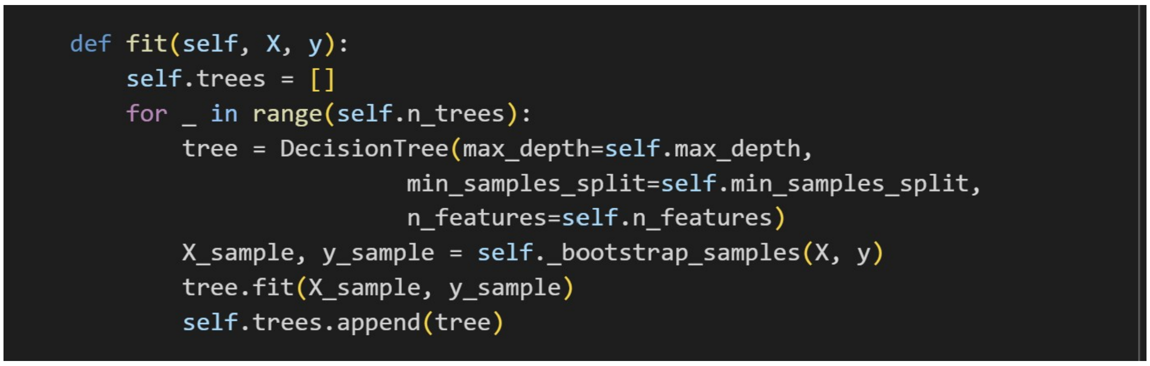
Attributes:

n trees: The number of decision trees in the random forest. max depth: The maximum depth allowed for each decision tree. min samplessplit: The min- imum number of samples required to split a node in each decision tree. n features: The number of features to consider when looking for the best split in each decision tree. trees: A list to store the individual decision trees that make up the random forest.

Methods:

init (): This is the constructor method that initializes the attributes of the Random-Forest object. It takes in the parameters mentioned above and assigns them to the corresponding attributes.

In summary, this code defines a basic skeleton for a random forest model, with options to specify the number of trees, maximum depth, minimum sam- ples required for splitting, and the number of features to consider for each split. However, the code does not include any functionality for training or using the random forest. Additional methods would need to be implemented to complete the random forest model, such as methods for fitting the model to training data and making predictions. self.trees = []: Initializes an empty list to store the decision trees that will be created during training. for i in range(self.n trees):: Iterates self.n trees times, where self.n trees is a parameter specifying the number of trees in the ensemble. tree = DecisionTree(...): Creates an instance of a decision tree called a tree. The parameters max depth, min samples split, and n features are passed to the decision tree constructor. These



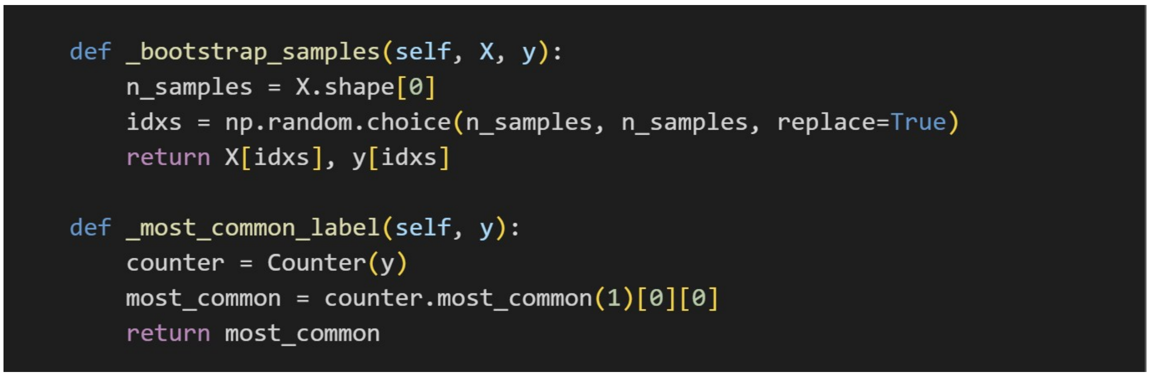
```
def fit(self, X, y):
    self.trees = []
    for _ in range(self.n_trees):
        tree = DecisionTree(max_depth=self.max_depth,
                            min_samples_split=self.min_samples_split,
                            n_features=self.n_features)
        X_sample, y_sample = self._bootstrap_samples(X, y)
        tree.fit(X_sample, y_sample)
        self.trees.append(tree)
```

Figure 5.12: Fit Function

parameters control the maximum depth of the tree, the minimum number of samples required to split a node, and the number of features to consider when looking for the best split, respectively.

X sample, y sample = self.bootstrap samples(X, y): Generates a random bootstrap sample from the original dataset X and corresponding labels y. This random sampling with replacement creates a new dataset and label set of the same size as the original, allowing for variability in the training data for each tree.

tree.fit(X sample, y sample): Trains the decision tree tree using the bootstrap sample X sample and y sample. self.trees.append(tree): Adds the trained decision tree tree to the list of trees in the ensemble.



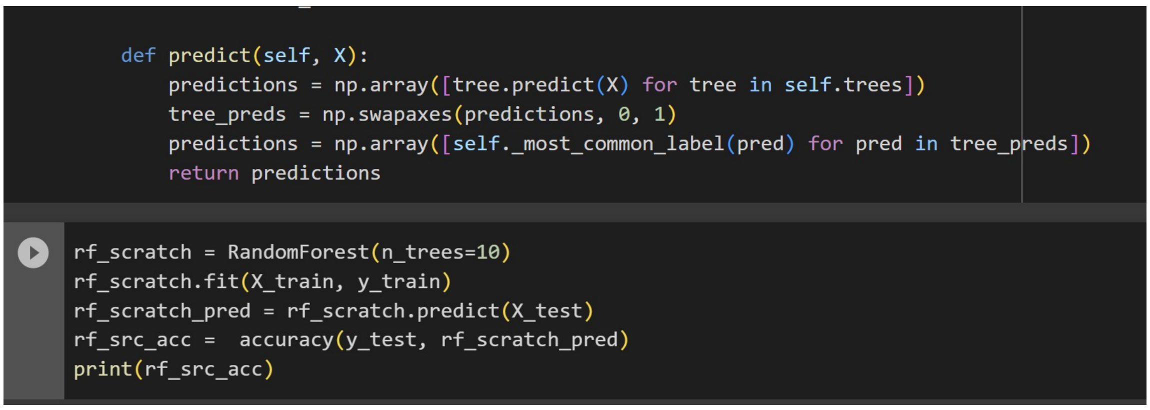
```
def _bootstrap_samples(self, X, y):
    n_samples = X.shape[0]
    idxs = np.random.choice(n_samples, n_samples, replace=True)
    return X[idxs], y[idxs]

def _most_common_label(self, y):
    counter = Counter(y)
    most_common = counter.most_common(1)[0][0]
    return most_common
```

Figure 5.13: Bootstrap Function

bootstrap samples(self, X, y):

This method takes two input arrays, X and y, representing a dataset of samples and corresponding labels. It determines the number of samples (n samples) in the dataset by accessing the shape of X. It generates random indices (idxs) by using np.random.choice function, which randomly selects n sample indices from the range of indices in the dataset. The replace=True parameter allows for replacement, meaning the same index can be selected multiple times. It returns a subset of the input arrays, X and y, using the random indices obtained. This subset contains randomly selected samples with replacements. most common label(self, y):

This method takes a single input array y, which represents the labels of a dataset. It uses the Counter class from the collections module to count the occurrences of each unique label in y. It retrieves the most common label by calling the most common(1) on the Counter object, which returns a list of tuples. The most common label is accessed by indexing [0][0]. It returns the most common label in the dataset.



```
    def predict(self, X):
        predictions = np.array([tree.predict(X) for tree in self.trees])
        tree_preds = np.swapaxes(predictions, 0, 1)
        predictions = np.array([self._most_common_label(pred) for pred in tree_preds])
        return predictions
```

```
rf_scratch = RandomForest(n_trees=10)
rf_scratch.fit(X_train, y_train)
rf_scratch_pred = rf_scratch.predict(X_test)
rf_src_acc =  accuracy(y_test, rf_scratch_pred)
print(rf_src_acc)
```

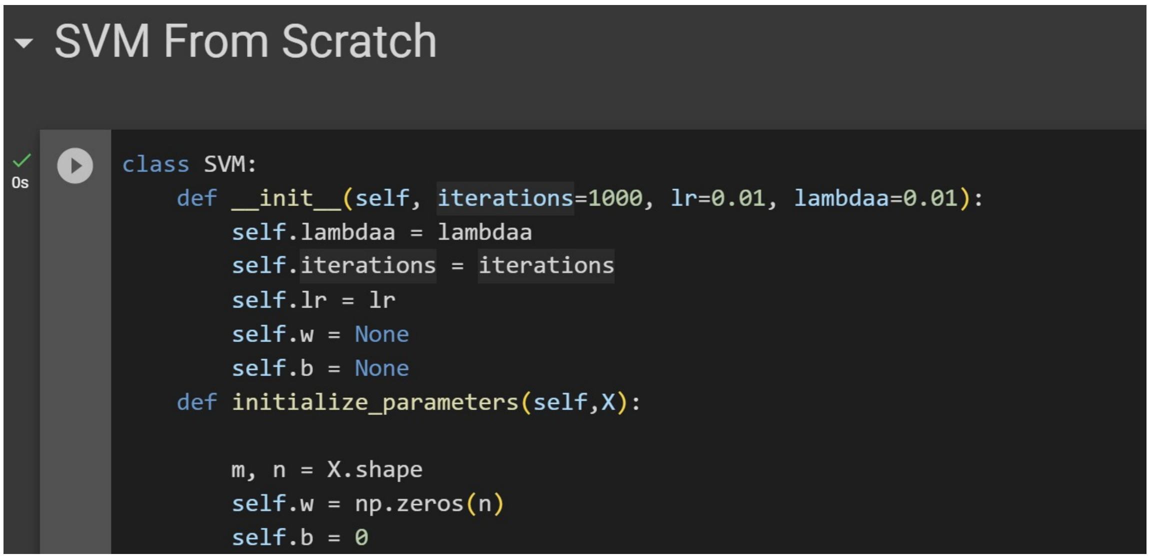
Figure 5.14: Predict

It takes an input X as the data to be predicted. It initializes an empty array called predictions to store the predictions made by individual decision trees. It iterates through each tree in the self.trees list and predict the output for the input X using that tree. The predicted values are stored in the predictions array. It swaps the axes of the predictions array using np.swapaxes to rearrange the predictions from a shape of (n trees, n samples) to (n samples, n trees).

It initializes another array called tree preds to store the rearranged predictions. It iterates through each prediction in tree preds and uses the most common label method to determine the most common prediction for each sample. The resulting predictions are stored in the predictions array.

Finally, it returns the predictions array containing the aggregated predictions from all the decision trees.

5.4.3 Support Vector Machine


SVM From Scratch

```
class SVM:
    def __init__(self, iterations=1000, lr=0.01, lambdaa=0.01):
        self.lambdaa = lambdaa
        self.iterations = iterations
        self.lr = lr
        self.w = None
        self.b = None
    def initialize_parameters(self,X):

        m, n = X.shape
        self.w = np.zeros(n)
        self.b = 0
```

Figure 5.15: Support Vector Machine Implemtation

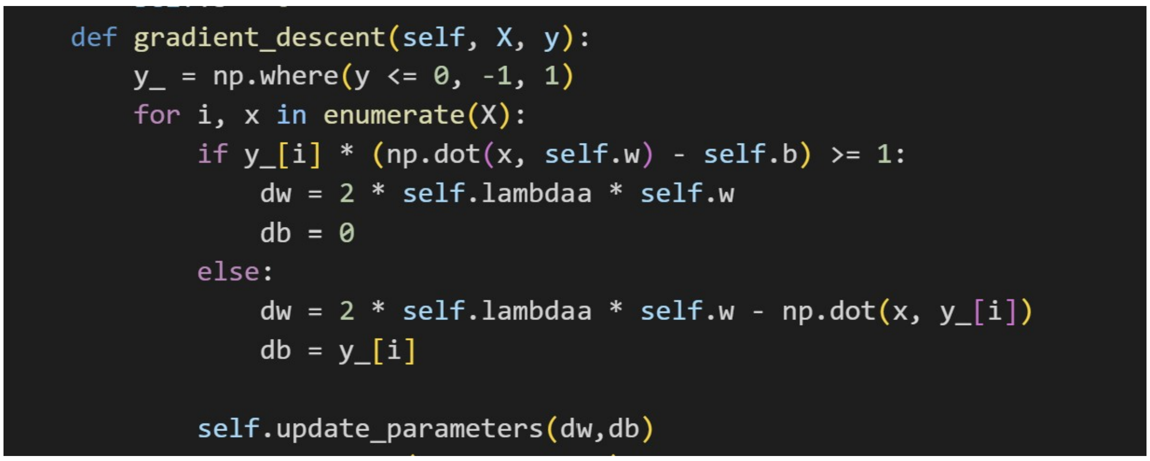
The init method is the constructor of the class. It initializes various parameters such as iterations, lr (learning rate), and lambdaa (regularization parameter). The default values are set to 1000 iterations, 0.01 learning rate, and 0.01 regularization parameter.

The initialize parameters method takes a matrix X as input and initializes the weight vector w and bias term b with zeros. The dimensions of w depend on the number of features in the input data.

A typical implementation of SVM involves training the model using optimization techniques such as gradient descent or quadratic programming to find the optimal hyperplane that separates different classes in the data The function takes two parameters: X (the input features) and y (the target labels). It creates a modified target variable y by replacing all values of y less than or equal to 0 with -1, and the rest with 1. This is done to convert the binary labels into -1 and 1 for convenience in the subsequent calculations.

It iterates over each input feature x in the dataset X using the enumerate function, which provides the index i along with the corresponding value.

Inside the loop, it checks if the current sample is correctly classified or not by com-

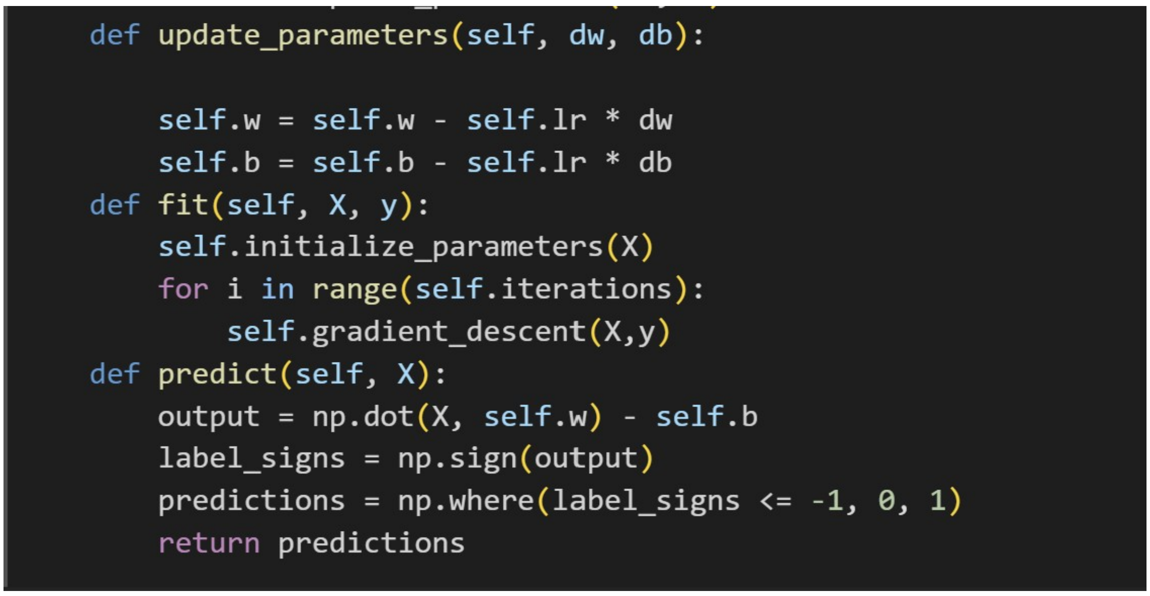


```
def gradient_descent(self, X, y):
    y_ = np.where(y <= 0, -1, 1)
    for i, x in enumerate(X):
        if y_[i] * (np.dot(x, self.w) - self.b) >= 1:
            dw = 2 * self.lambdaa * self.w
            db = 0
        else:
            dw = 2 * self.lambdaa * self.w - np.dot(x, y_[i])
            db = y_[i]

        self.update_parameters(dw,db)
```

Figure 5.16: Gradient Decent

paring y[i] multiplied by the difference between the dot product of x and the weight vector self.w and the bias term self.b with 1. If the condition is true (correctly classified), it updates the weight and bias gradients as follows: dw is set to 2 * self.lambdaa * self.w, where self.lambdaa is a regularization parameter. db is set to 0 (no update). If the sample is misclassified, it updates the weight and bias gradients as follows:

dw is set to 2 * self.lambdaa * self.w - np.dot(x, y[i]), where np.dot(x, y[i]) represents the dot product between the input feature x and the corresponding label y[i]. db is set to y[i] (the corresponding label). After calculating the weight and bias gradients, it calls the update parameters method to update the weights and biases using these gradients.



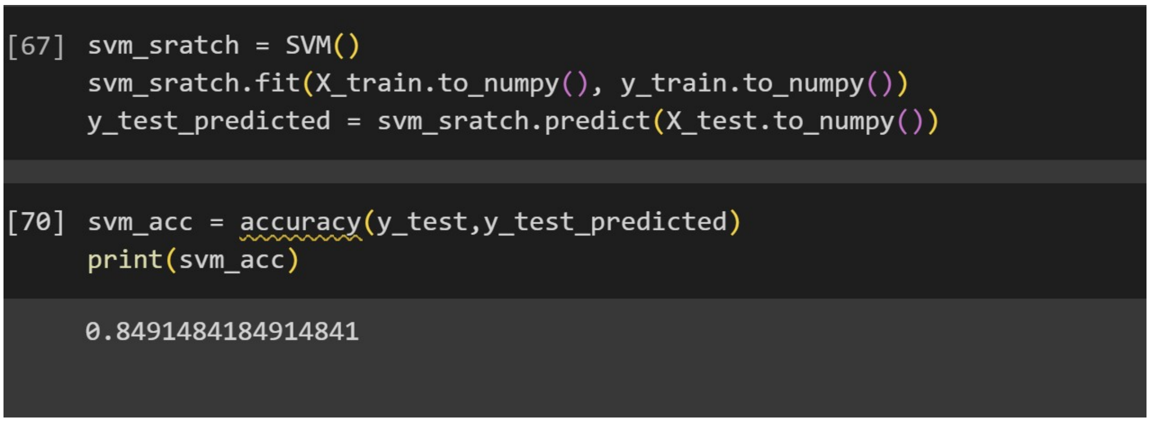
```
    def update_parameters(self, dw, db):

        self.w = self.w - self.lr * dw
        self.b = self.b - self.lr * db
    def fit(self, X, y):
        self.initialize_parameters(X)
        for i in range(self.iterations):
            self.gradient_descent(X,y)
    def predict(self, X):
        output = np.dot(X, self.w) - self.b
        label_signs = np.sign(output)
        predictions = np.where(label_signs <= -1, 0, 1)
        return predictions
```

Figure 5.17: Update Paraneter

update parameters(self, dw, db): This method updates the model's parameters (weights w and bias b) using the computed gradients (dw and db) and the learning rate (lr). It subtracts the product of the learning rate and the gradients from the current parameter values.

fit(self, X, y): This method initializes the parameters and then iterates a specified number of times (iterations). In each iteration, it calls the gradient descent() method (not shown) to compute the gradients and updates the parameters using update parameters().

predict(self, X): This method performs predictions on new data X. It computes the dot product between the input data and the weights (self.w), subtracts the bias (self.b), and obtains the output values. Then, it assigns a label of either 0 or 1 to each output based on a threshold of -1. If the output is less than or equal to -1, it assigns 0; otherwise, it assigns 1. The resulting predictions are returned.



```
[67] svm_sratch = SVM()
     svm_sratch.fit(X_train.to_numpy(), y_train.to_numpy())
     y_test_predicted = svm_sratch.predict(X_test.to_numpy())

[70] svm_acc = accuracy(y_test,y_test_predicted)
     print(svm_acc)

     0.8491484184914841
```

Figure 5.18: Calling SVM Functions

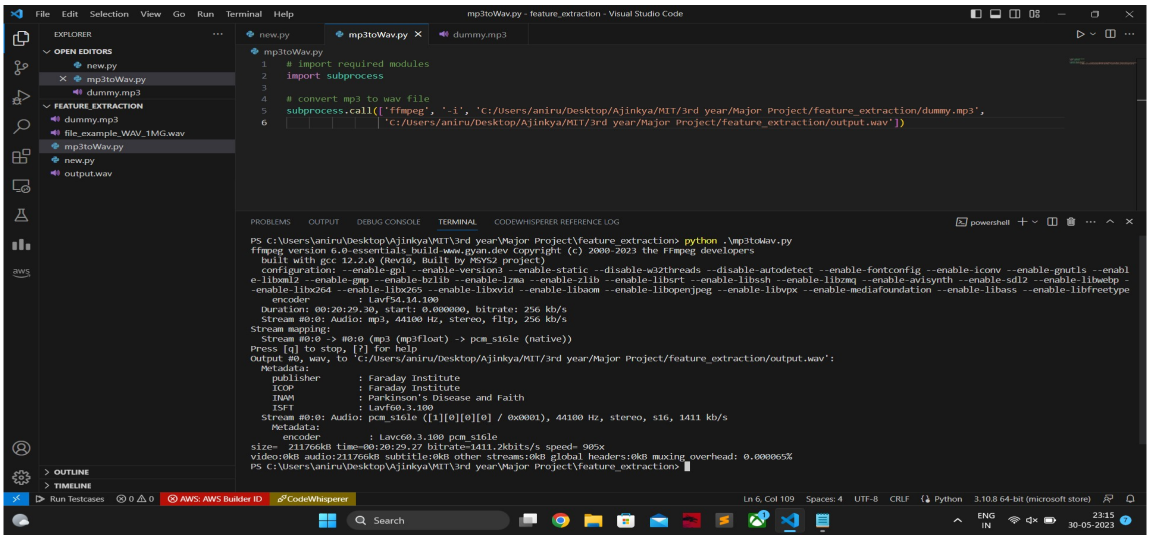
svm scratch = SVM(): This line creates an instance of the SVM class, which is presumably a custom implementation of the SVM algorithm. svm scratch.fit(X train.to numpy(), y train.to numpy()): This line trains the SVM classifier using the fit method. It takes the training data X train (presumably a feature matrix) and y train (presumably the corresponding target values) and fits the SVM model to the data.

y test predicted = svm scratch.predict(X test.to numpy()): This line predicts the target values for the test data X test using the trained SVM model. The predict method takes the test data and returns the predicted target values y test predicted.

svm acc = accuracy(y test, y test predicted): This line calculates the accuracy of the SVM model by comparing the predicted target values y test predicted with the true target values y test. The accuracy function (not shown in the provided code) is likely a custom implementation that calculates the accuracy metric.

print(svm acc): Finally, this line prints the calculated accuracy of the SVM model on the test data.

5.4.4 Convert .mp3 to .wav



```python
# import required modules
import subprocess

# convert mp3 to wav file
subprocess.call(['ffmpeg', '-i', 'C:/Users/aniru/Desktop/Ajinkya/MIT/3rd year/Major Project/feature_extraction/dummy.mp3',
                 'C:/Users/aniru/Desktop/Ajinkya/MIT/3rd year/Major Project/feature_extraction/output.wav'])
```

```
PS C:\Users\aniru\Desktop\Ajinkya\MIT\3rd year\Major Project\feature_extraction> python .\mp3toWav.py
ffmpeg version 6.0-essentials_build-www.gyan.dev Copyright (c) 2000-2023 the FFmpeg developers
  built with gcc 12.2.0 (Rev10, Built by MSYS2 project)
  configuration: --enable-gpl --enable-version3 --enable-static --disable-w32threads --disable-autodetect --enable-fontconfig --enable-iconv --enable-gnutls --enabl
e-libxml2 --enable-gmp --enable-bzlib --enable-lzma --enable-zlib --enable-libsrt --enable-libssh --enable-libzmq --enable-avisynth --enable-sdl2 --enable-libwebp -
-enable-libx264 --enable-libx265 --enable-libxvid --enable-libaom --enable-libopenjpeg --enable-libvpx --enable-mediafoundation --enable-libass --enable-libfreetype
    encoder         : Lavf54.14.100
  Duration: 00:20:29.30, start: 0.000000, bitrate: 256 kb/s
  Stream #0:0: Audio: mp3, 44100 Hz, stereo, fltp, 256 kb/s
Stream mapping:
  Stream #0:0 -> #0:0 (mp3 (mp3float) -> pcm_s16le (native))
Press [q] to stop, [?] for help
Output #0, wav, to 'C:/Users/aniru/Desktop/Ajinkya/MIT/3rd year/Major Project/feature_extraction/output.wav':
  Metadata:
    publisher       : Faraday Institute
    ICOP            : Faraday Institute
    INAM            : Parkinson's Disease and Faith
    ISFT            : Lavf60.3.100
  Stream #0:0: Audio: pcm_s16le ([1][0][0][0] / 0x0001), 44100 Hz, stereo, s16, 1411 kb/s
    Metadata:
      encoder         : Lavc60.3.100 pcm_s16le
size=  211766kB time=00:20:29.27 bitrate=1411.2kbits/s speed= 905x
video:0kB audio:211766kB subtitle:0kB other streams:0kB global headers:0kB muxing overhead: 0.000065%
PS C:\Users\aniru\Desktop\Ajinkya\MIT\3rd year\Major Project\feature_extraction>
```

Figure 5.19: Convert .mp3 to .wav

Explanation:

Importing the required module:

1. **Importing the required module:** This line im- ports the subprocess module, which allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes.

2. **Converting MP3 to WAV file:** This line uses the subprocess.call() function to execute a command in the command-line shell. In this case, the command is 'ffmpeg', which is a command-line tool for handling multimedia files. The arguments passed to ffmpeg are as follows:

   i specifies the output file path and name for the WAV file that will be generated. When this code is executed, it calls the ffmpeg command-line tool and passes the specified arguments to it. ffmpeg reads the input MP3 file and converts it to a WAV file with the specified output path and name.

   This code snippet demonstrates how to use the subprocess module to execute a command-line tool (ffmpeg) from within a Python script to convert an MP3 file to a WAV file.

Chapter 6

Implementation

The following steps will be used to implement the problem statement described above.

1. **Data Cleaning and Forecasting Process Flow:** Start by obtaining the raw data for Parkinson's disease. This could include medical records, clinical assessments, sensor data, or any other relevant sources. Perform data cleaning, which involves removing any noise, outliers, or irrelevant information from the raw data. This step ensures that the data is in a usable format for further analysis. Preprocess the cleaned data to transform it into a format that is suitable for forecasting. This may involve normalization, scaling, or encoding categorical variables. Create an architectural diagram that outlines the flow of data from the raw data to the final forecast. This diagram should illustrate the preprocessing steps and how the data is fed into the forecasting model.

2. **Data Preprocessing:** Once the raw data is obtained, it needs to be preprocessed to make it understandable and suitable for analysis. Apply techniques such as data cleaning, missing value imputation, and outlier detection to handle any data quality issues. Normalize or scale the data if needed to ensure that features are on a similar scale and to improve the performance of machine learning algorithms. Encode categorical variables using techniques like one-hot encoding or label encoding if necessary.

3. **Train-Test Data Split:** Split the preprocessed dataset into two parts: a training set and a testing set. The training set will be used to train the machine learning algorithms, while the testing set will be used to evaluate the performance of the trained models. The typical split ratio is 70-30 or 80-20, where 70 percent or 80 percent of the data is used for training and the remaining percentage is used for testing.

4. **Machine Learning Model Training:** Apply machine learning algorithms such as Support Vector Machines (SVM), k-nearest Neighbors (KNN), and Random Forest to train the Parkinson's disease data. These algorithms are commonly used for classification tasks and can provide insights into the presence or absence of Parkinson's disease based on the input features. Train each algorithm using the training data and tuning the hyperparameters to achieve the best performance.

5. **Model Testing:** Once the models are trained, it's time to evaluate their performance using the testing set. For each trained model (SVM, KNN, and Random Forest), apply it to the testing data and obtain the predicted outputs. Compare the predicted outputs with the actual labels in the testing set to assess the accuracy and performance of each model. Calculate additional evaluation metrics such as precision, recall, and F1 score to get a comprehensive understanding of the model's performance. Precision represents the proportion of true positive predictions out of all positive predictions, while recall represents the proportion of true positive predictions out of all actual positive instances. The F1 score is the harmonic mean of precision and recall and provides a balanced measure of the model's performance.

6. **Comparison of Classification Accuracy:**

   Once the models have been tested and evaluated, it's time to compare their classification accuracy to determine which one performs better for Parkinson's disease detection. Calculate the accuracy of each model by dividing the number of correctly classified instances by the total number of instances in the testing set. Compare the accuracy of the SVM, KNN, and Random Forest models to identify the one with the highest accuracy. It's important to consider not only the accuracy but also other evaluation metrics like precision, recall, and F1 score to have a comprehensive understanding of the model's performance. Additionally, consider the computational complexity and interpretability of each model to assess their practicality for real-world applications.

6.1 Libraries and Modules

1. **numpy:** NumPy is a fundamental library for numerical computing in Python. It provides a powerful n-dimensional array object, ndarray, which allows efficient manipula- tion of large arrays and matrices. NumPy offers a wide range of mathematical functions for array operations, including mathematical operations, linear al- gebra, random number generation, and more. It is widely used in scientific computing, data analysis, and machine learning due to its performance and efficiency.

2. **pandas:** Pandas is a popular library for data manipulation and analysis in Python. It provides data structures like Series (1-dimensional) and DataFrame (2-dimensional) to handle structured data efficiently. Pandas offers functionalities for reading and writing data in various formats, data cleaning and preprocessing, data alignment, merging, reshaping, and data aggregation. It is widely used for tasks like data cleaning, exploration, transformation, and analysis in fields such as data science, finance, and social sciences.

3. **matplotlib:** Matplotlib is a comprehensive plotting library in Python for creating static, animated, and interactive visualizations. It provides a flexible and user-friendly interface to create a wide range of plots, including line plots, scatter plots, bar plots, histograms, pie charts, and more. Matplotlib offers extensive customization options for plots, such as labels, titles, colors, line styles, markers, and annotations. It is highly customizable and can be used in various environments, including Jupyter notebooks, GUI applications, and web applications.

4. **sklearn:** Scikit-learn is a powerful machine-learning library in Python, that provides a range of algorithms and tools for machine-learning tasks. It offers

functionalities for classification, regression, clustering, dimensionality reduction, model selection, and preprocessing. Scikit-learn provides a consistent API and supports a wide range of machine learning models, including decision trees, support vector machines, random forests, neural networks, and more. It also includes utilities for model evaluation, cross-validation, hyperparameter tuning, and pipeline construction. Scikit-learn is widely used in academia and industry for machine learning projects and research.

5. **collections:** The collections module is part of the Python standard library and provides additional data structures beyond the built-in types. It includes specialized container datatypes that offer alternatives to the built-in collections like lists, tuples, and dictionaries. Notable data structures in the collections module include defaultdict, OrderedDict, Counter, and deque. defaultdict provides a dictionary with a default value for missing keys, OrderedDict preserves the order of insertion, Counter counts the occurrences of elements, and deque is a double-ended queue with efficient append and pop operations. These data structures are useful in various scenarios, such as counting elements, creating ordered dictionaries, implementing queues, and solving algorithmic problems efficiently.

6. **warnings:** The warnings module in Python provides a way to handle warnings that occur during program execution. Here's a short description of the warnings module: The warnings module allows you to control the behavior of warnings in Python programs. It defines various warning categories, such as DeprecationWarning and SyntaxWarning, representing different types of issues or potential problems. With the warnings module, you can modify the default warning handling behavior, such as whether to display warnings or treat them as errors. You can use functions like filterwarnings(), showwarning(), resetwarnings(), and simple- filter() to customize the handling of warnings.

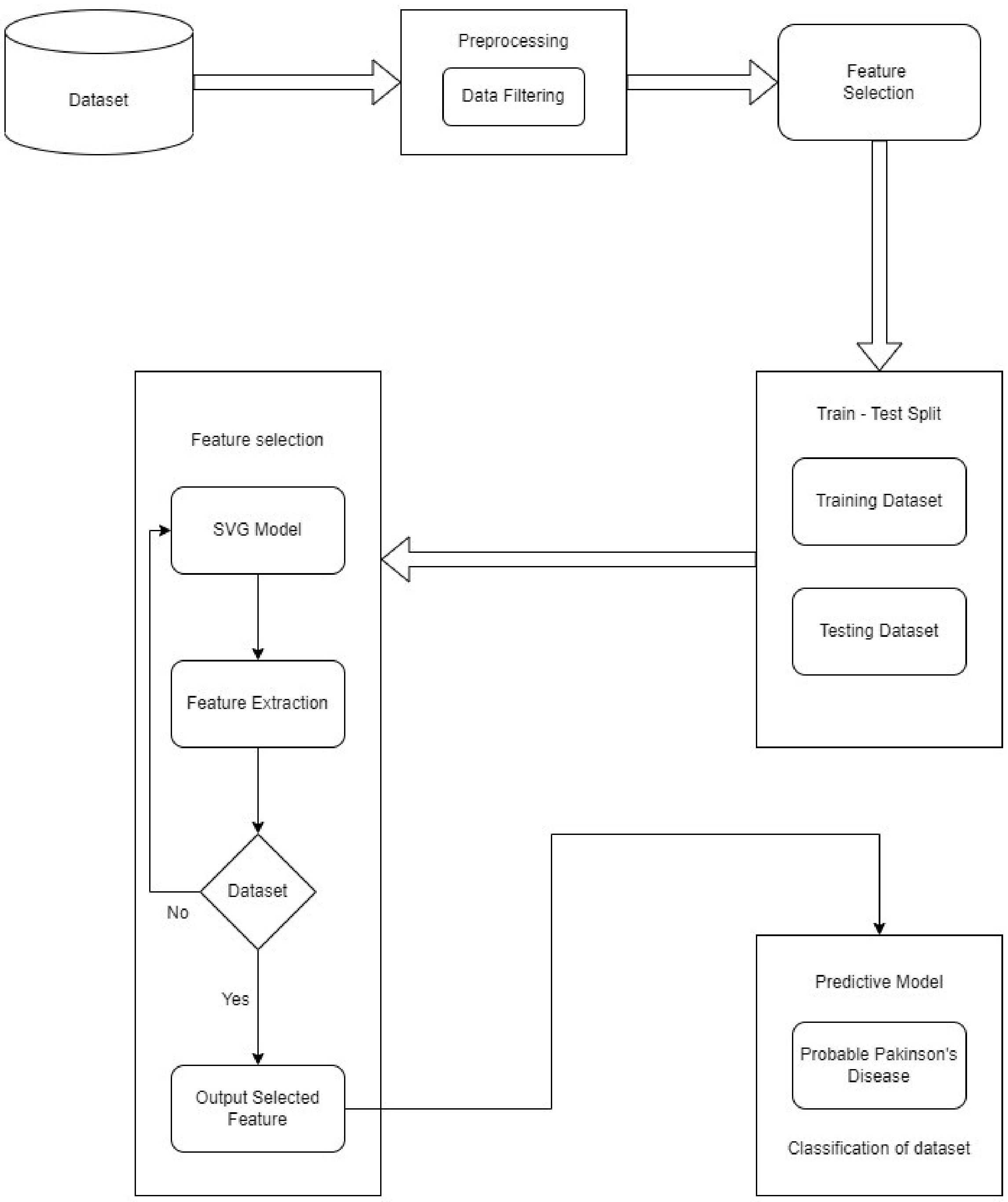


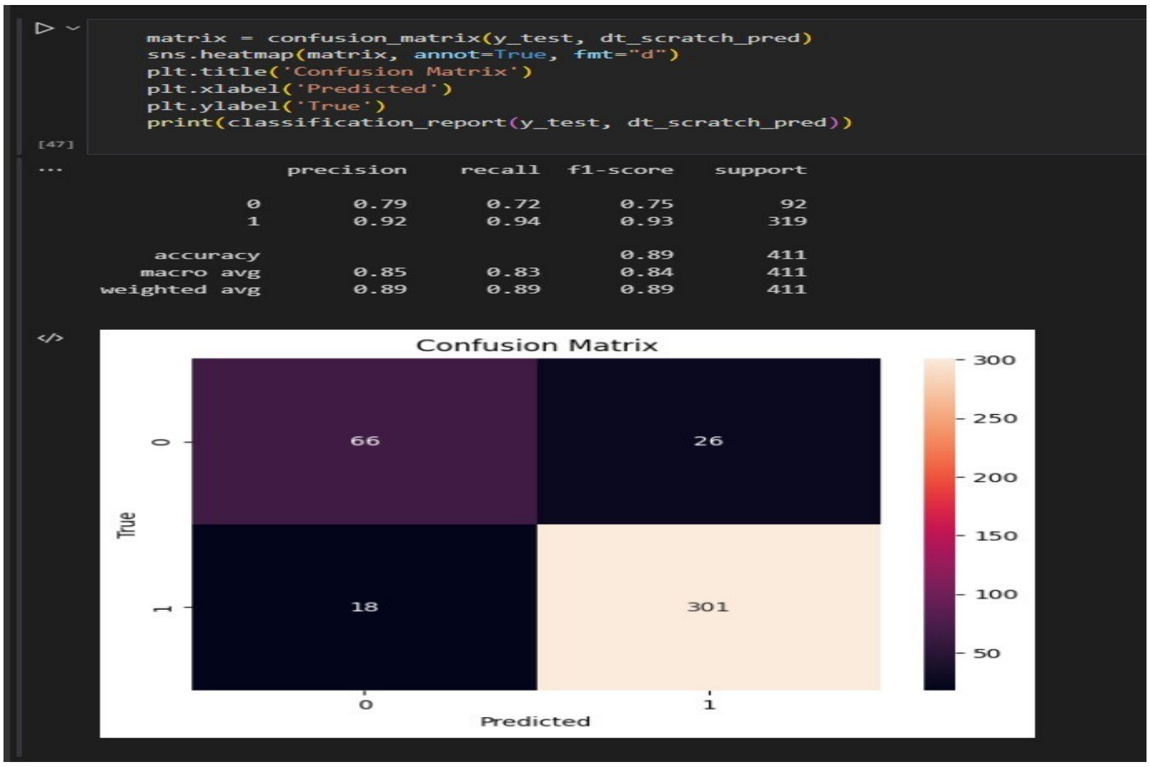
Figure 6.1: Flow Chart

Chapter 7

Result Analysis/Performance Evaluation

7.1 Confusion Matrix

7.1.1 Decision Tree Confusion Matrix



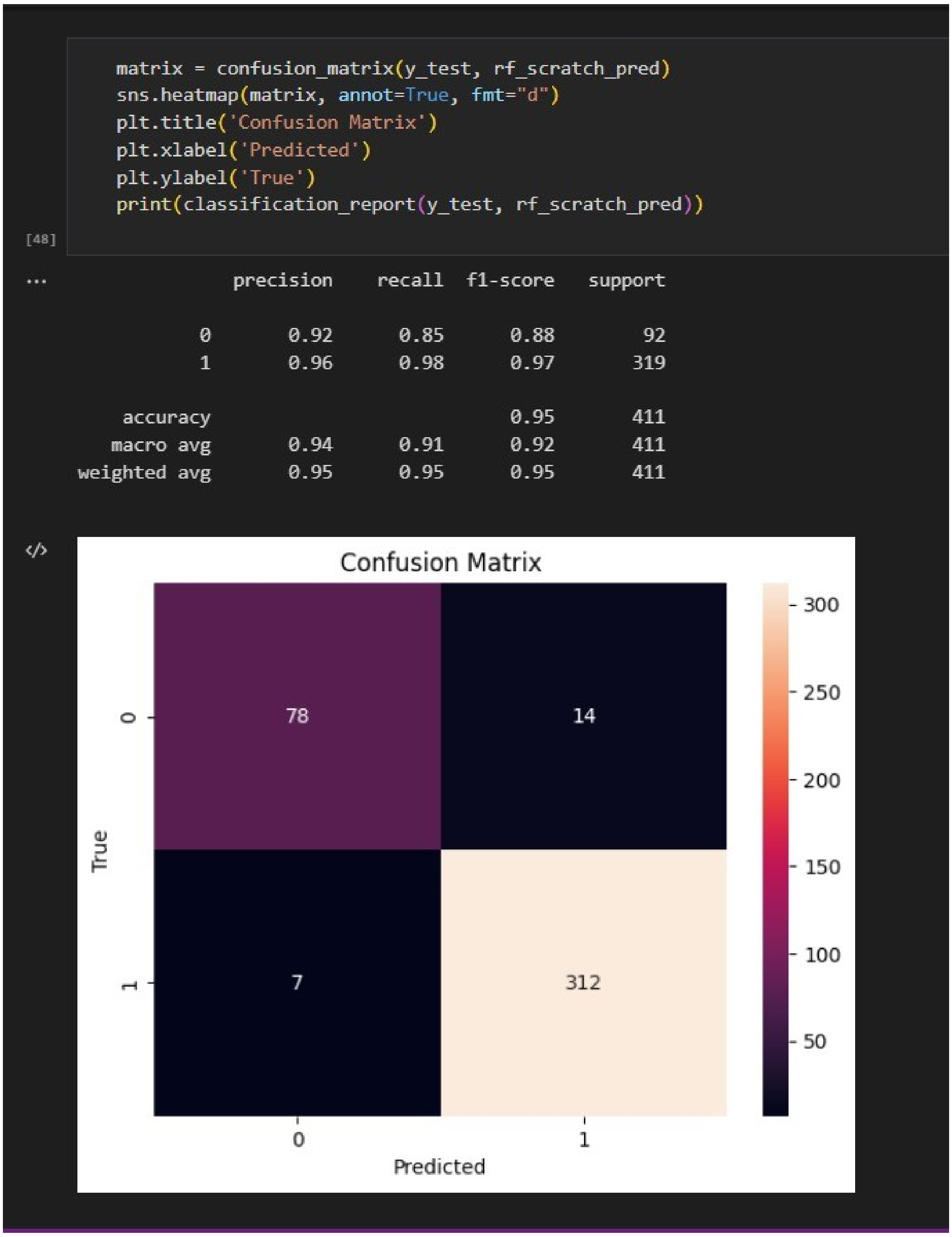
```
matrix = confusion_matrix(y_test, dt_scratch_pred)
sns.heatmap(matrix, annot=True, fmt="d")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
print(classification_report(y_test, dt_scratch_pred))
```

[47]

```
              precision    recall  f1-score   support

           0       0.79      0.72      0.75        92
           1       0.92      0.94      0.93       319

    accuracy                           0.89       411
   macro avg       0.85      0.83      0.84       411
weighted avg       0.89      0.89      0.89       411
```

Figure 7.1: Decision Tree Confusion Matrix

7.1.2 Random Forest Confusion Matrix



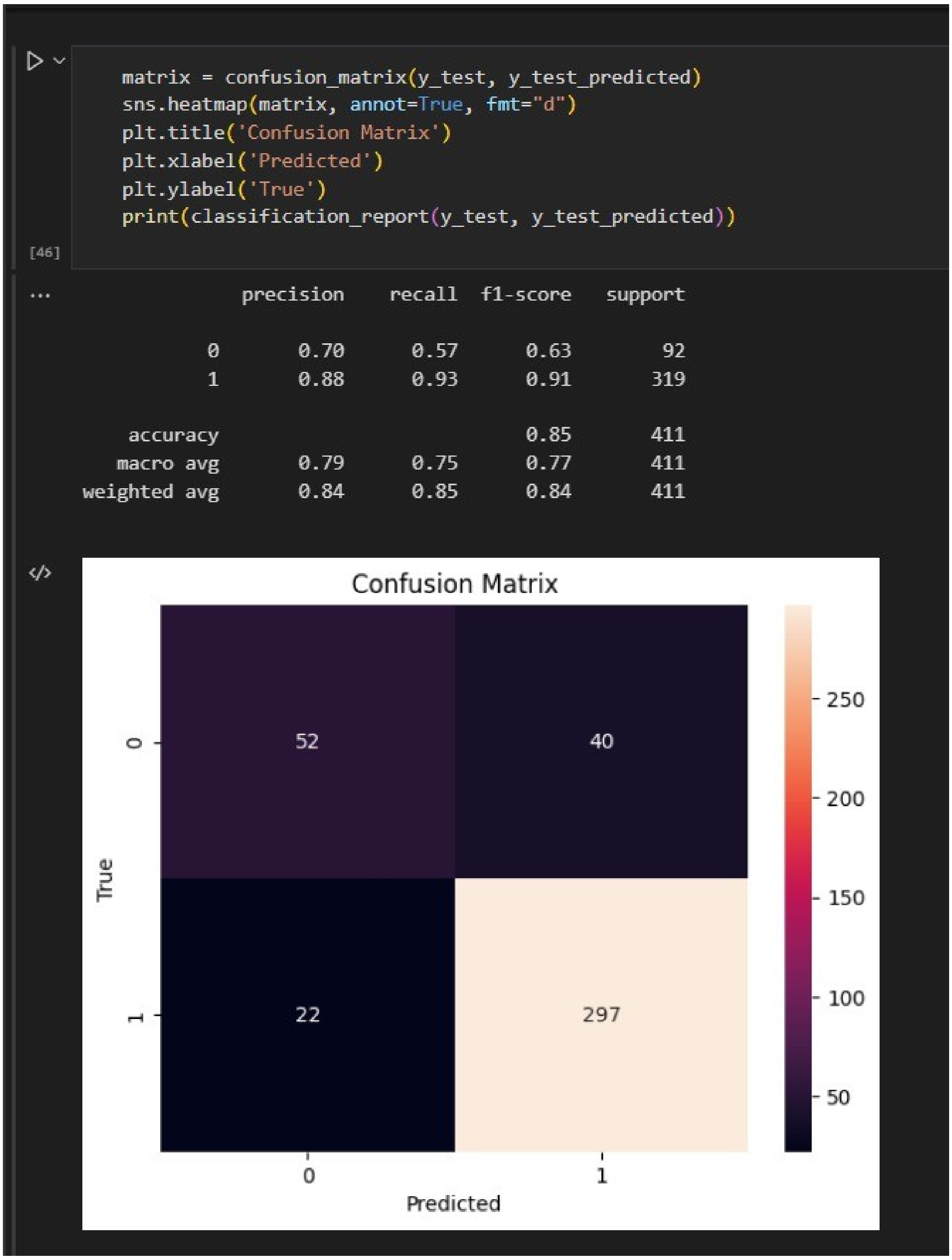
```
matrix = confusion_matrix(y_test, rf_scratch_pred)
sns.heatmap(matrix, annot=True, fmt="d")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
print(classification_report(y_test, rf_scratch_pred))
```

[48]

```
              precision    recall  f1-score   support

           0       0.92      0.85      0.88        92
           1       0.96      0.98      0.97       319

    accuracy                           0.95       411
   macro avg       0.94      0.91      0.92       411
weighted avg       0.95      0.95      0.95       411
```

Figure 7.2: Random Forest Confusion Matrix

7.1.3 Support Vector Machine Confusion Matrix



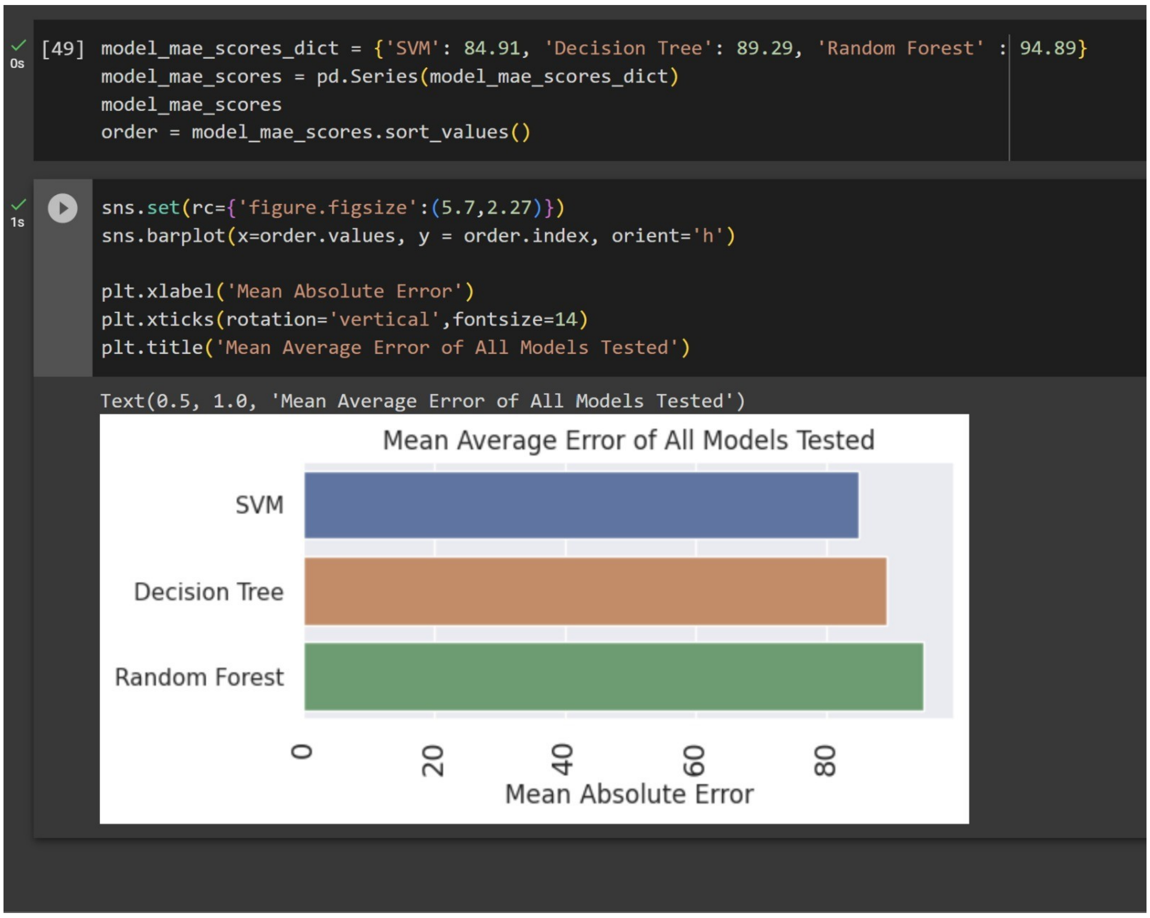
```
matrix = confusion_matrix(y_test, y_test_predicted)
sns.heatmap(matrix, annot=True, fmt="d")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
print(classification_report(y_test, y_test_predicted))
```

[46]

```
              precision    recall  f1-score   support

           0       0.70      0.57      0.63        92
           1       0.88      0.93      0.91       319

    accuracy                           0.85       411
   macro avg       0.79      0.75      0.77       411
weighted avg       0.84      0.85      0.84       411
```

Figure 7.3: Support Vector Machine Confusion Matrix

7.2 Observations and Results



```
[49] model_mae_scores_dict = {'SVM': 84.91, 'Decision Tree': 89.29, 'Random Forest' : 94.89}
     model_mae_scores = pd.Series(model_mae_scores_dict)
     model_mae_scores
     order = model_mae_scores.sort_values()
```

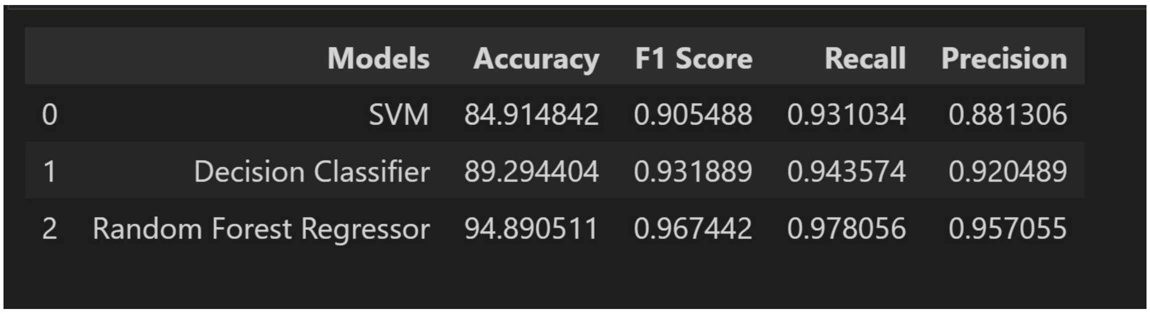
```
sns.set(rc={'figure.figsize':(5.7,2.27)})
sns.barplot(x=order.values, y = order.index, orient='h')

plt.xlabel('Mean Absolute Error')
plt.xticks(rotation='vertical',fontsize=14)
plt.title('Mean Average Error of All Models Tested')
```

```
Text(0.5, 1.0, 'Mean Average Error of All Models Tested')
```

Figure 7.4: Model Comparison

Bagging, short for Bootstrap Aggregating, is a popular ensemble learning method used in machine learning. In bagging, multiple models are trained independently on different subsets of the training data, and their predictions are combined to make the final prediction. The bagging method is utilized in the Random Forest algorithm, which is one of the models you provided. Here are the advantages of bagging: Reduction of Variance: Bagging helps reduce the variance of the model by training multiple models on different subsets of the data. Each model captures different aspects of the data, and by combining their predictions, the overall variance is reduced. This can lead to more robust and stable predictions. Handling Overfitting: Bagging can effectively handle overfitting, which occurs when a model learns the training data too well and fails to generalize to unseen data. By training each model on a different subset of the data, bagging introduces diversity and reduces the chances of overfitting. The models in the ensemble collectively learn from the data's overall patterns rather than memorizing individual instances.

Improved Prediction Accuracy: Bagging can lead to improved prediction accuracy compared to using a single model. By combining the predictions from multiple models, the ensemble can capture a broader range of patterns and make more accurate predictions. This can be particularly beneficial when dealing with complex datasets or datasets with high variance. Robustness to Outliers: Bagging is robust to outliers in the training data. Since each model is trained on a different subset of the data, outliers may have less influence on the overall predictions. The averaging or voting mechanism used to combine the predictions tends to mitigate the impact of outliers, making the ensemble more robust. It helps improve the prediction accuracy, handle overfitting, and provide robust predictions for Parkinson's disease detection.



	Models	Accuracy	F1 Score	Recall	Precision
0	SVM	84.914842	0.905488	0.931034	0.881306
1	Decision Classifier	89.294404	0.931889	0.943574	0.920489
2	Random Forest Regressor	94.890511	0.967442	0.978056	0.957055

Figure 7.5: Result Analysis

SVM (Support Vector Machine): Accuracy: 84.91F1 Score: 0.9055 Recall: 0.9310 Precision: 0.8813 The SVM al- gorithm is a powerful and widely used classification technique. It aims to find an optimal hyperplane that separates data points of different classes with the maximum margin. SVMs perform well in scenarios where there is a clear separation between classes and a small number of outliers. However, in this particular comparison, the SVM model achieved relatively lower accuracy, F1 Score, recall, and precision com- pared to the other two models.

Decision Tree Classifier: Accuracy: 89.29F1 Score: 0.9319 Recall: 0.9436 Precision: 0.9205 Decision trees are intuitive and interpretable models that partition the feature space based on decision rules. They are capable of capturing complex relationships between features and the target variable. The Decision Tree Classifier outperformed the SVM in terms of accuracy, F1 Score, recall, and precision, achieving higher values for all metrics. This suggests that the Decision Tree model provides a better representation of the underlying patterns in the dataset.

Random Forest Regressor: Accuracy: 94.89F1 Score: 0.9674 Recall: 0.9781 Precision: 0.9571 Random Forest is an ensemble learning method that combines multiple decision trees to make pre- dictions. It uses bagging and feature randomization to improve generalization and reduce overfitting. The Random Forest Regressor achieved the highest values for all performance metrics, including accuracy, F1 Score, recall, and precision. This indicates that the Random Forest model was able to capture the complex relation- ships in the dataset effectively and make accurate predictions for Parkinson's disease detection.

Based on the provided results, the Random Forest Regressor appears to be the most beneficial algorithm for Parkinson's disease detection in this comparison. It demonstrated the highest accuracy, F1 Score, recall, and precision among the three models. The Random Forest's ability to handle complex relationships, reduce overfitting, and produce accurate predictions makes it a promising choice for this task. However, it's essential to consider other factors such as computational complexity, interpretability, and the specific requirements of the problem before making a final decision.

Chapter 8

Conclusion

8.1 Conclusion

Research into Parkinson's disease is now of utmost importance, and an early diagnosis can enhance the quality of life for the patient. The techniques for speech analysis have lately made enormous strides. Early detection of PD is essential to better understanding the causes of the condition, launching therapy approaches, and enabling the development of efficient drugs. The study assesses how different aural environments affect various PD detection techniques. The findings show that classification work is significantly impacted by background noise. To frequently monitor the neurological status, it must be properly taken into consideration.

This study's major goal is to show how speech signal analysis may be used to identify Parkinson's disease. Due to the non-intrusive nature of voice measures, speech processing has long shown that it offers great potential for PD diagnosis. This study's objective is to assess how well different categorization techniques work. Several classifiers will be evaluated on a speech dataset using statistical analysis and visualization, and different assessment criteria will be contrasted.

8.2 Future Scope

1. **User-Friendly Interface:** Develop a user-friendly interface for the website, ensuring that it is intuitive, accessible, and easy to navigate. Consider incorporating modern design principles and responsive layouts to provide an optimal user experience across various devices.

2. **Online Assessment:** Create an online assessment form or questionnaire that collects relevant information from users. Include questions about symptoms, medical history, and any other factors that can aid in Parkinson's disease detection.

3. **Data Collection:** Implement a secure and efficient system to collect and store user data. Ensure compliance with data protection regulations and prioritize user privacy and confidentiality.

4. **Sensor Integration:** Explore the integration of sensors or wearable devices to capture additional data for Parkinson's disease detection. For example, accelerometer data from smartphones or smartwatches can be used to analyze movement patterns.

5. **Machine Learning Models:** Incorporate the trained machine learning models for Parkinson's disease detection into the website. These models can analyze user-provided data, such as symptoms and sensor data, to generate predictions or probabilities of Parkinson's disease presence.

6. **Real-Time Results:** Design the website to provide real-time results or feedback to users after they submit their data. This can help users gain insights into their potential risk of Parkinson's disease and prompt them to seek further medical evaluation.

7. **Educational Resources:** Offer educational resources, such as articles, videos, or infographics, on the website to educate users about Parkinson's disease, its symptoms, risk factors, and available treatments. Ensure that the information is accurate, up-to-date, and easily understandable.

8. **Referral to Healthcare Professionals:** Provide clear instructions and recommendations for users based on their assessment results. Encourage users to consult with healthcare professionals or specialists for a formal diagnosis and appropriate medical guidance.

9. **Collaboration with Medical Experts:** Establish collaborations with healthcare professionals or medical experts who specialize in Parkinson's disease. Seek their input and expertise to ensure the accuracy and reliability of the website's content and assessment methodology.

10. **Continuous Improvement:** Regularly update and improve the website based on user feedback, advancements in technology, and new research in Parkinson's disease detection. Incorporate new features or functionalities to enhance the user experience and accuracy of the assessment.

Appendices

Appendix A

Plagiarism Report of Text

Detection Of Parkinson's Disease Through Speech Analysis

by Piyush Bhondave

Detection Of Parkinson’s Disease Through Speech Analysis

ORIGINALITY REPORT

14%	9%	13%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Abdelaziz Testas. "Distributed Machine Learning with PySpark", Springer Science and Business Media LLC, 2023 Publication	2%
2	Dr. Shitalkumar R. Sukhdeve, Sandika S. Sukhdeve. "Chapter 2 Google Colaboratory", Springer Science and Business Media LLC, 2023 Publication	1%
3	www.mdpi.com Internet Source	1%
4	www.nature.com Internet Source	1%
5	vegibit.com Internet Source	1%
6	medium.com Internet Source	<1%
7	Lecture Notes in Computer Science, 2007. Publication	<1%

8	www.openacessjournal.com Internet Source	<1 %
9	acris.aalto.fi Internet Source	<1 %
10	scholarworks.rit.edu Internet Source	<1 %
11	ebin.pub Internet Source	<1 %
12	Nossayba Darraz, Ikram Karabila, Anas El-Ansari, Nabil Alami, Mohamed Lazaar, Mostafa El Mallahi. "Using Sentiment Analysis to Spot Trending Products", 2023 Sixth International Conference on Vocational Education and Electrical Engineering (ICVEE), 2023 Publication	<1 %
13	studygyaan.com Internet Source	<1 %
14	logicmojo.com Internet Source	<1 %
15	science.vu.nl Internet Source	<1 %
16	www.geeksforgeeks.org Internet Source	<1 %
17	www.researchsquare.com	

	Internet Source	<1 %
18	Vikas Khare, Ankita Jain. "Predict the Performance of Driverless Car through the Cognitive Data Analysis and Reliability Analysis based Approach", e-Prime - Advances in Electrical Engineering, Electronics and Energy, 2023 Publication	<1 %
19	babu73140.medium.com Internet Source	<1 %
20	riunet.upv.es Internet Source	<1 %
21	"Next Generation Systems and Networks", Springer Science and Business Media LLC, 2023 Publication	<1 %
22	cheatography.com Internet Source	<1 %
23	link.springer.com Internet Source	<1 %
24	www.semanticscholar.org Internet Source	<1 %
25	opus4.kobv.de Internet Source	<1 %

26	sigport.org Internet Source	<1%
27	ijiemr.org Internet Source	<1%
28	info.statik.uni-due.de Internet Source	<1%
29	sami.fel.cvut.cz Internet Source	<1%
30	webthesis.biblio.polito.it Internet Source	<1%
31	Antonio Alcántara, Carlos Ruiz. "On data-driven chance constraint learning for mixed-integer optimization problems", Applied Mathematical Modelling, 2023 Publication	<1%
32	Ton Duc Thang University Publication	<1%

Exclude quotes	On	Exclude matches	< 1 words
Exclude bibliography	On		

References

Appakaya, S. B., & Sankar, R. (2018). Classification of parkinson's disease using pitch synchronous speech analysis. In *2018 40th annual international conference of the ieee engineering in medicine and biology society (embc)* (p. 1420-1423). doi: 10.1109/EMBC.2018.8512481

Arias-Vergara, T., Vasquez-Correa, J., Orozco-Arroyave, J., Klumpp, P., & Nöth, E. (2018). Unobtrusive monitoring of speech impairments of parkinson's disease patients through mobile devices. In *2018 ieee international conference on acoustics, speech and signal processing (icassp)* (p. 6004-6008). doi: 10.1109/ICASSP.2018.8462332

Gunduz, H. (2019). Deep learning-based parkinson's disease classification using vocal feature sets. *IEEE Access*, *7*, 115540-115551. doi: 10.1109/ACCESS.2019.2936564

Laganas, C., Iakovakis, D., Hadjidimitriou, S., Charisis, V., Dias, S. B., Bostantzopoulou, S., ... Hadjileontiadis, L. J. (2022). Parkinson's disease detection based on running speech data from phone calls. *IEEE Transactions on Biomedical Engineering*, *69*(5), 1573-1584. doi: 10.1109/TBME.2021.3116935

Narendra, N., Schuller, B., & Alku, P. (2021). The detection of parkinson's disease from speech using voice source information. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *29*, 1925-1936. doi: 10.1109/TASLP.2021.3078364

Orozco, J. R., Vasquez, J., Hoenig, F., Arias-Londoño, J. D., Vargas-Bonilla, J., Skodda, S., ... Noeth, E. (2016, 03). Towards an automatic monitoring of the neurological state of parkinson's patients from speech.. doi: 10.1109/ICASSP.2016.7472927

Quan, C., Ren, K., & Luo, Z. (2021, 01). A deep learning based method for parkinson's disease detection using dynamic features of speech. *IEEE Access*, *9*, 10239-10252. doi: 10.1109/ACCESS.2021.3051432

Sakar, C. O., Serbes, G., Gunduz, A., Tunc, H. C., Nizam, H., Sakar, B. E., ... Apaydin, H. (2019). A comparative analysis of speech signal processing algorithms for parkinson's disease classification and the use of the tunable q-factor wavelet transform. *Applied Soft Computing*, *74*, 255-263. Retrieved from https://www.sciencedirect.com/science/article/pii/S1568494618305799 doi: https://doi.org/10.1016/j.asoc.2018.10.022

Vásquez-Correa, J. C., Serrà, J., Orozco-Arroyave, J. R., Vargas-Bonilla, J. F., & Nöth, E. (2017). Effect of acoustic conditions on algorithms to detect parkinson's disease from speech. In *2017 ieee international conference on acoustics, speech and signal processing (icassp)* (p. 5065-5069). doi: 10.1109/ICASSP.2017.7953121

Wang, W., Lee, J., Harrou, F., & Sun, Y. (2020). Early detection of parkinson's disease using deep learning and machine learning. *IEEE Access*, *8*, 147635-147646. doi: 10.1109/ACCESS.2020.3016062