INDUSTRIAL ORIENTED MINI PROJECT

Report

On

DOWN SYNDROME DETECTION USING ADVANCED MACHINE LEARNING ALGORITHM

Submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

In

INFORMATION TECHNOLOGY

By

Karangula Yeshwanth Reddy - 22261A1232

Karthikeya Reddy Velagala - 22261A1233

Under the guidance of

Dr. D. Vijaya Lakshmi

Professor and HOD, Department of IT

DEPARTMENT OF INFORMATION TECHNOLOGY

MAHATMA GANDHI INSTITUTE OF TECHNOLOGY
(AUTONOMOUS)

(Affiliated to JNTUH, Hyderabad; Eight UG Programs Accredited by NBA; Accredited by NAAC with 'A++' Grade)

Gandipet, Hyderabad, Telangana, Chaitanya Bharati (P.O), RangaReddy District, Hyderabad– 500075, Telangana

2024-2025

CERTIFICATE

This is to certify that the **Industrial Oriented Mini Project** entitled **DOWN SYNDROME DETECTION USING ADVANCED MACHINE LEARNING ALGORITHM** submitted by **Karangula Yeshwanth Reddy (22261A1232), Karthikeya Reddy Velagala (22261A1233)** in partial fulfillment of the requirements for the Award of the Degree of Bachelor of Technology in Information Technology as specialization is a record of the bona fide work carried out under the supervision of **Dr. D. Vijaya Lakshmi**, and this has not been submitted to any other University or Institute for the award of any degree or diploma.

Internal Supervisor:

Dr. D. Vijaya Lakshmi
Professor and HOD
Dept. of IT

IOMP Supervisor:

Dr. U. Chaitanya
Assistant Professor
Dept. of IT

EXTERNAL EXAMINAR

Dr. D. Vijaya Lakshmi
Professor and HOD
Dept. of IT

DECLARATION

We hear by declare that the **Industrial Oriented Mini Project** entitled **DOWN SYNDROME DETECTION USING ADVANCED MACHINE LEARNING ALGORITHMS** is an original and bona fide work carried out by us as a part of fulfilment of Bachelor of Technology in Information Technology, Mahatma Gandhi Institute of Technology, Hyderabad, under the guidance of **Dr. D. Vijaya Lakshmi ,**Professor and HOD Department of IT, MGIT.

No part of the project work is copied from books /journals/ internet and wherever the portion is taken, the same has been duly referred in the text. The report is based on the project work done entirely by us and not copied from any other source.

Karangula Yeshwanth Reddy - 22261A1232

Karthikeya Reddy Velagala - 22261A1233

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success. They have been a guiding light and source of inspiration towards the completion of the **Industrial Oriented Mini Project**.

We would like to express our sincere gratitude and indebtedness to our Internal supervisor **Dr. D. Vijaya Lakshmi,** Professor and HOD , Dept. of IT, who has supported us throughout our project with immense patience and expertise.

We are also thankful to our honorable Principal of MGIT **Prof. G. Chandramohan Reddy** and **Dr. D. Vijaya Lakshmi**, Professor and HOD, Department of IT, for providing excellent infrastructure and a conducive atmosphere for completing this **Industrial Oriented Mini Project** successfully.

We are also extremely thankful to our IOMP supervisor **Dr. U. Chaitanya**, Assistant Professor, Department of IT, and senior faculty **Mrs. B. Meenakshi** Department of IT for their valuable suggestions and guidance throughout the course of this project.

We convey our heartfelt thanks to the lab staff for allowing us to use the required equipment whenever needed.

Finally, we would like to take this opportunity to thank our families for their support all through the work. We sincerely acknowledge and thank all those who gave directly or indirectly their support for the completion of this work.

Karangula Yeshwanth Reddy - 22261A1232

Karthikeya Reddy Velagala -22261A1233

ABSTRACT

Down Syndrome (DS), or trisomy 21, is a congenital genetic disorder characterized by the presence of an extra chromosome 21. It leads to developmental delays, intellectual disabilities, and various health complications. Traditional diagnostic methods, including karyotyping and non-invasive prenatal testing (NIPT), while effective, are either time-consuming, expensive, or inaccessible in resource-limited settings. With the advent of high-throughput technologies such as microarrays, vast gene expression data can be leveraged to develop computational models for rapid, scalable, and cost-effective diagnosis.

This project proposes an end-to-end machine learning pipeline for the prediction of Down Syndrome using gene expression profiles sourced from public datasets (GSE6408 and GSE9321) in the Gene Expression Omnibus (GEO). Raw .gpr files were processed to extract meaningful features such as normalized signal intensities (e.g., CH1 and CH2 signal and background medians), with poor-quality spots filtered based on flag values. The pipeline incorporates key preprocessing steps, including missing value handling, synthetic minority oversampling (SMOTE) for class imbalance, noise injection for robustness, and standardization using StandardScaler.

Several machine learning models—Random Forest, XGBoost, and LightGBM—were trained and evaluated for classification accuracy. The Random Forest model demonstrated superior performance and was deployed alongside the scaler in a web application using Flask. The app enables users to upload microarray-derived CSV files, automatically processes the data, and returns predictions on whether each sample indicates the presence of Down Syndrome.

This project highlights the feasibility and effectiveness of integrating genomic data analysis with machine learning for the early detection of genetic disorders. It underscores the potential of AI-driven biomedical tools in aiding clinical decisions, particularly in genomics-based diagnostics

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 MOTIVATION

Down Syndrome, also known as Trisomy 21, is one of the most common genetic disorders, occurring in approximately 1 out of every 700 live births. It is caused by the presence of an extra copy of chromosome 21 and leads to developmental delays, intellectual disabilities, and physical abnormalities. Detecting this condition early during pregnancy is crucial, as it allows parents and healthcare providers to prepare for medical care, support, and interventions that can significantly improve the child's quality of life.

Traditional diagnostic techniques such as karyotyping, fluorescence in situ hybridization (FISH), and amniocentesis, although accurate, are often invasive, expensive, and time-consuming. These methods may not be readily available in all regions and can pose risks to both the fetus and the mother. Moreover, with increasing awareness and demand for safer, faster, and non-invasive diagnostic solutions, there is a growing need to explore alternatives that leverage technological advancements.

With the rapid development of bioinformatics and artificial intelligence, machine learning (ML) offers a powerful approach to analyze complex biological data such as gene expression profiles. These datasets, however, are high-dimensional and non-linear, making them difficult to interpret using traditional statistical methods. Advanced ML algorithms like Random Forest, XGBoost, and LightGBM can automatically learn patterns from such data, improving both accuracy and efficiency in classification tasks.

The motivation behind this project lies in utilizing these advanced ML techniques to create a non-invasive, highly accurate, and scalable system for detecting Down Syndrome using gene expression data. This not only reduces the dependency on invasive diagnostic procedures but also supports early and reliable decision-making in clinical settings. Ultimately, the goal is to combine medical insight with computational intelligence to improve health outcomes for individuals affected by this genetic disorder.

.

1.2 PROBLEM STATEMENT

Down syndrome (Trisomy 21) is a common genetic disorder caused by an extra copy of chromosome 21, leading to developmental delays and intellectual disabilities. Current diagnostic methods like karyotyping and invasive prenatal testing (e.g., amniocentesis) are either costly, risky, or time-consuming. There is a growing need for a non-invasive, accurate, and efficient approach to detect Down syndrome early during pregnancy. The challenge lies in analyzing high-dimensional genetic data effectively to identify biomarkers associated with Trisomy 21. This project aims to apply advanced machine learning techniques—such as XGBoost, LightGBM, and Random Forest—to gene expression datasets for early and accurate detection of Down syndrome, improving diagnostic accuracy and aiding in timely intervention.

1.3 EXISTING SYSTEM

Current methods for detecting Down syndrome rely heavily on clinical and cytogenetic techniques such as karyotyping, FISH (Fluorescence In Situ Hybridization), and non-invasive prenatal testing (NIPT). These approaches focus on identifying the presence of an extra chromosome 21 in the fetus, which is the genetic basis of Down syndrome. While accurate, these procedures are often labor-intensive, time-consuming, and costly. In some research settings, gene expression data is also used, where statistical methods like t-tests, ANOVA, or principal component analysis (PCA) are applied to find differentially expressed genes between affected and control groups. However, these traditional analytical techniques often struggle to model the complex, high-dimensional, and non-linear patterns present in genetic datasets, and they require extensive domain knowledge to interpret results effectively.

.

1.3.1 Limitations

- **Invasiveness and Accessibility of Traditional Methods:** Current clinical techniques such as karyotyping, FISH, and amniocentesis are either invasive, time-consuming, or expensive. These methods require skilled professionals, advanced laboratory infrastructure, and are not always accessible in low-resource or rural healthcare settings.
- **Limited Handling of Complex Data:** Statistical methods like t-tests, ANOVA, and PCA used in gene expression analysis struggle to model the complex, high-dimensional, and

non-linear nature of genetic data. These methods may miss intricate patterns that are crucial for accurate diagnosis.

- **High Dependence on Domain Expertise:** Traditional approaches often require significant bioinformatics knowledge to select features, tune parameters, and interpret results. This dependence on expert input can introduce human bias and limits the scalability of the methods in real-world clinical workflows.
- **Cost and Time Constraints:** Cytogenetic and prenatal screening procedures, although accurate, involve high operational costs and long processing times, delaying early diagnosis and intervention.
- **Limited Generalization in Population Diversity:** Current methods may not generalize well across genetically diverse populations, potentially leading to reduced accuracy and increased false positives or negatives when applied outside well-studied cohorts.

1.4 PROPOSED SYSTEM

The proposed system aims to enhance the accuracy and efficiency of Down syndrome detection by applying advanced machine learning algorithms to gene expression data. Instead of relying solely on manual statistical analysis or cytogenetic observations, this system utilizes algorithms such as Random Forest, XGBoost, and LightGBM to automatically classify samples based on gene expression profiles. These models are capable of handling large, high-dimensional datasets, managing missing values, and learning complex relationships between genes without human bias. The proposed system enables faster diagnosis, higher accuracy, and scalability to larger datasets. Moreover, it offers the potential for identifying key genetic biomarkers associated with Down syndrome, supporting not only classification but also further biological insights. This shift from manual interpretation to intelligent automation represents a significant advancement in the field of genetic disorder detection.

1.4.1 ADVANTAGES

- **High Accuracy with Advanced Algorithms:** By utilizing robust machine learning algorithms like Random Forest, XGBoost, and LightGBM, the system achieves high accuracy in classifying Down Syndrome based on gene expression data. These models are capable of capturing complex, non-linear relationships within high-dimensional datasets, outperforming traditional statistical methods.

- **Non-Invasive Diagnosis Support:** The proposed system analyzes gene expression data, enabling a non-invasive approach to support Down Syndrome detection. This reduces dependency on costly and risky procedures like amniocentesis or karyotyping, offering a safer alternative for early diagnosis.

- **Automated and Scalable Analysis:** Machine learning models automate the feature selection and classification processes, reducing the need for manual interpretation and domain-specific expertise. This enhances scalability and consistency, making it feasible to deploy the system in real-world clinical environments.

- **Faster Decision-Making:** The system provides rapid analysis and output once the gene expression data is input, supporting timely diagnosis and intervention. This speed is particularly valuable during prenatal screening, where quick decisions can significantly impact outcomes.

- **Potential for Biomarker Discovery:** In addition to classification, the models can identify important features (genes) that contribute to Down Syndrome detection. This offers valuable insights for researchers and can contribute to further understanding of the genetic basis of the disorder.

1.5 OBJECTIVES

- **To Develop a Machine Learning-Based Detection System:** Design and implement a predictive system using advanced machine learning algorithms (Random Forest,

XGBoost, and LightGBM) for accurate classification of Down Syndrome based on gene expression profiles.

- **To Utilize Public Gene Expression Datasets:** Integrate and preprocess gene expression data from publicly available datasets (GSE6408 for Down Syndrome and GSE9321 for Control) to train and validate the detection models.

- **To Enhance Diagnostic Accuracy and Speed:** Leverage the computational power of machine learning to improve the accuracy, efficiency, and speed of Down Syndrome diagnosis compared to conventional statistical or cytogenetic methods.

- **To Create a Non-Invasive Diagnostic Approach:** Offer a safer alternative to traditional invasive procedures by using gene expression analysis as a non-invasive diagnostic support tool.

- **To Enable User-Friendly Predictions via a Web Interface:** Develop a simple and intuitive web-based frontend that allows users to upload gene expression data (CSV) and receive instant classification results—"Down Syndrome" or "Not Down Syndrome."

1.6 HARDWARE AND SOFTWARE REQUIREMENTS

Software Requirements

- **Software:**

  The Down Syndrome prediction system is developed using **Python IDLE** as the integrated development environment (IDE). Python IDLE provides a simple and effective environment for writing, testing, and debugging Python code required for machine learning model training and inference.

- **Primary Language:**

  The project is implemented primarily in **Python**, which is ideal for machine learning, bioinformatics data processing, and web development. Key Python libraries used include **scikit-learn**, **LightGBM**, **XGBoost**, **pandas**, **NumPy**, **Matplotlib**, **Seaborn**,

GEOparse, and **imblearn** to handle data processing, model training, visualization, and imbalanced dataset handling.

- **Frontend Tools:**

  The frontend is developed using standard web technologies:

  - **HTML** for page structure,
  - **CSS** for styling and layout,
  - **JavaScript** to enable dynamic interactivity.

- **IDE Environment:**

  The coding and debugging are primarily done within **Python IDLE**, which supports seamless integration of all Python libraries needed for the project.

Hardware Requirements

- **Processor:**

  An **Intel i5** processor or equivalent is recommended to efficiently handle the computational requirements of gene expression data processing, model training, and prediction inference.

- **RAM:**

  A minimum of **8 GB RAM** is required to smoothly run machine learning algorithms and handle multitasking during development and inference. For enhanced performance, **16 GB RAM** is preferred.

- **Storage:**

  The system requires at least **100 GB SSD** storage space to accommodate the Python environment, libraries, datasets such as GSE6408 and GSE9321, model files, and project outputs.

2. LITERATURE SURVEY

[1] **Gupta et al., 2020**

"Machine Learning Applications in Intellectual and Developmental Disabilities," ML in Intellectual & Developmental Disabilities Research. This study explores machine learning models inspired by neurological processes to predict developmental disabilities. The models offer interpretability and potential cross-domain applicability, though a lack of Down Syndrome-specific datasets limits accuracy.

[2] **Raza et al., 2021**

"Transfer Learning Facial Analysis for Down Syndrome Diagnosis," Transfer Learning Facial Analysis Conference. Raza et al. apply deep learning to facial image analysis, achieving high diagnostic performance by detecting facial features characteristic of Down Syndrome. However, broader dataset diversity is needed to improve model robustness.

[3] **Cibrian et al., 2019**

"Limitations in Speech Recognition for Developmental Disabilities," Speech Recognition Limitation Study. This research focuses on speech recognition techniques aimed at communication enhancement. While promising, models face performance issues in noisy environments and require adaptation for real-time applications.

[4] **Çelik et al., 2017**

"Machine Learning Techniques in Gene Estimation," 25th Signal Processing and Communications Applications Conference (SIU).
Çelik et al. present efficient gene estimation methods using machine learning with decent accuracy but note challenges in generalizing and scaling models.

[5] **Li et al., 2019**

"Cascaded Machine Learning Framework for Imbalanced Data," IEEE Access. The authors propose a cascaded machine learning system handling imbalanced datasets effectively with accurate predictions and feature correlation management. The framework's complexity and computational cost are potential drawbacks.

[6] **Ramanathan et al., 2018**

"Probabilistic Machine Learning Techniques for Prediction," ICACCI – International Conference on Advances in Computing, Communications and Informatics. This paper explores probabilistic machine learning to improve prediction confidence. The study points out the need for real-time implementations and more extensive dataset validation.

[7] **Rodrigues et al., 2019**

"Imaging-based Machine Learning Analysis," Neuroimaging Review. Rodrigues et al. review imaging-based ML approaches, providing valuable medical imaging insights. However, an automated classification pipeline is absent, indicating a research gap.

Table 2.1 Literature Survey of Research papers

S. No	Author year of publication	Journal / Conference	Method / Approach	Merits	Demerits / Research Gaps
1	Çelik et al., 2017	25th Signal Processing and Communications Applications Conference (SIU)	Machine Learning Techniques	Efficient gene estimation approach with decent accuracy	Lack of model generalization and scalability
2	Li et al., 2019	IEEE Access	Cascaded ML framework for imbalanced data	Accurate predictions with feature correlation handling	Complex design and high computational cost
3	Ramanathan et al., 2018	ICACCI – Int'l Conf. on Advances in Computing, Communications and Informatics	Probabilistic ML techniques	Probabilistic modeling improves prediction confidence	Needs real-time implementation and dataset validation
4	Rodrigues et al.	Neuroimaging Review	Imaging-based ML analysis	Pictorial review offers medical imaging insights	Lacks automated classification pipeline
5	Gupta et al.	ML in Intellectual & Developmental Disabilities Research	Inspired by neurological ML models	Cross-domain applicability and interpretability	Need for dedicated Down syndrome-specific datasets
6	Raza et al.	Transfer Learning Facial Analysis	Deep learning & facial image analysis	High performance in diagnosis via facial cues	Needs diverse facial image datasets
7	Cibrian et al.	Speech Recognition Limitations Study	Speech recognition ML techniques	Focused on communication enhancement	Performance drops in noisy conditions and real-time usage

3. ANALYSIS AND DESIGN

The domain of genetic disorder prediction, particularly Down Syndrome, presents significant challenges due to the complexity of gene expression data and the need for early, accurate diagnosis. This project focuses on developing a robust predictive model to assist researchers and clinicians in identifying Down Syndrome using microarray gene expression datasets. By integrating multiple machine learning models, the system enhances the reliability and accuracy of predictions, potentially aiding in early screening and medical research.

At the core of this project lies the ability to collect and process gene expression profiles from public repositories such as GEO (Gene Expression Omnibus). In particular, datasets like **GSE6408** and **GSE9321** are used, which contain gene expression samples from both Down Syndrome patients and healthy controls. These datasets undergo preprocessing, normalization, and feature scaling to ensure high-quality input for the prediction models. This ensures that the system's outputs are based on clean, standardized, and biologically relevant data.

To achieve reliable prediction, the system employs a hybrid ensemble learning approach. It utilizes **Random Forest** to handle complex nonlinear relationships and reduce overfitting, **XGBoost** for gradient boosting and performance optimization, and **LightGBM** for its speed and efficiency in handling large, high-dimensional gene expression data. These models are trained and evaluated using standard machine learning metrics such as accuracy, confusion matrix, and classification report. Among them, the XGBoost model is selected and deployed due to its superior performance on this dataset.

This ensemble modeling approach ensures that the system captures a wide spectrum of gene expression patterns, enabling it to differentiate effectively between Down Syndrome and normal samples. The final model is serialized and integrated into a **Flask-based web application**, allowing users (such as researchers or clinicians) to upload a gene expression CSV file and receive an immediate prediction.

By combining domain-specific data, advanced ensemble learning techniques, and a user-friendly web interface, the system delivers a practical tool for aiding in Down Syndrome detection using genomic data.

3.1 MODULES

Data Acquisition Module

• **Functions:**

Retrieves gene expression data from publicly available repositories such as GEO (Gene Expression Omnibus). Extracts relevant biological features from .gpr, .csv, or .txt files, which include signal intensity, background noise, and spot quality. Parses metadata, sample types, and experimental conditions.

• **Input:**

Raw microarray data files or CSV files containing gene expression values and metadata such as sample ID, signal intensity, and spot quality flags.

• **Output:**

Structured datasets with cleaned and formatted gene expression values, filtered for quality, and ready for preprocessing.

Preprocessing Module

• **Functions:**

Cleans and normalizes raw gene expression data. Handles missing values, corrects background noise, filters out unreliable probes using flags, and extracts relevant statistical features. Optionally introduces Gaussian noise to increase data robustness and realism.

• **Input:**

Structured gene expression data with columns such as CH1_SIG_Median, CH1_BKD_Median, Flags, and VALUE.

• **Output:**

Cleaned, background-corrected, and normalized dataset suitable for machine learning modeling.

Data Balancing Module

• **Functions:**

Addresses class imbalance in the dataset using techniques such as SMOTE (Synthetic

Minority Over-sampling Technique). Ensures equal representation of Down syndrome and control samples in the training data.

• **Input:**

Imbalanced gene expression dataset with unequal numbers of Down syndrome and control samples.

• **Output:**

Balanced dataset with synthetic samples added for the minority class, improving classifier generalization.

Feature Scaling Module

• **Functions:**

Applies normalization techniques such as Standard Scaling or Min-Max Scaling to ensure all gene expression values are on the same scale, which is crucial for most machine learning models to perform efficiently.

• **Input:**

Balanced gene expression dataset with raw or preprocessed numerical values.

• **Output:**

Normalized dataset with scaled features, maintaining relative differences and reducing model bias due to magnitude differences.

Machine Learning Module

• **Functions:**

Trains predictive models using algorithms such as Random Forest, XGBoost, and LightGBM. Performs hyperparameter tuning and validation. Learns complex patterns from high-dimensional gene expression data to classify samples as Down syndrome or control.

• **Input:**

Preprocessed, balanced, and scaled dataset including both gene features and class labels.

• **Output:**

Trained machine learning model capable of classifying new gene expression samples. Includes performance metrics like accuracy, F1-score, and ROC-AUC.

Prediction Module

• **Functions:**

Applies the trained machine learning model to new user-uploaded gene expression samples. Generates predictions in the form of classification labels (Down syndrome or control) and optionally returns confidence scores or probabilities.

• **Input:**

CSV files containing gene expression values from unknown/test samples.

• **Output:**

Predicted class labels for each sample, along with probability scores or model confidence.

Web Interface Module

• **Functions:**

Provides a user interface for uploading gene expression files and viewing prediction results. Ensures ease of use for clinicians, researchers, and general users. Allows interaction with the system without requiring programming skills.

• **Input:**

User-provided .csv files uploaded through a web form.

• **Output:**

Prediction results displayed on screen, including classification (Down syndrome / control), probability/confidence, and optional feature contributions.

Backend API Module

• **Functions:**

Acts as the intermediary between frontend and machine learning logic. Handles requests, runs preprocessing pipeline, loads trained models, and returns predictions to the frontend.

• **Input:**

HTTP requests containing user-uploaded files or data.

• **Output:**

JSON responses with prediction labels and metadata about processing success or failure.

3.2 ARCHITECTURE

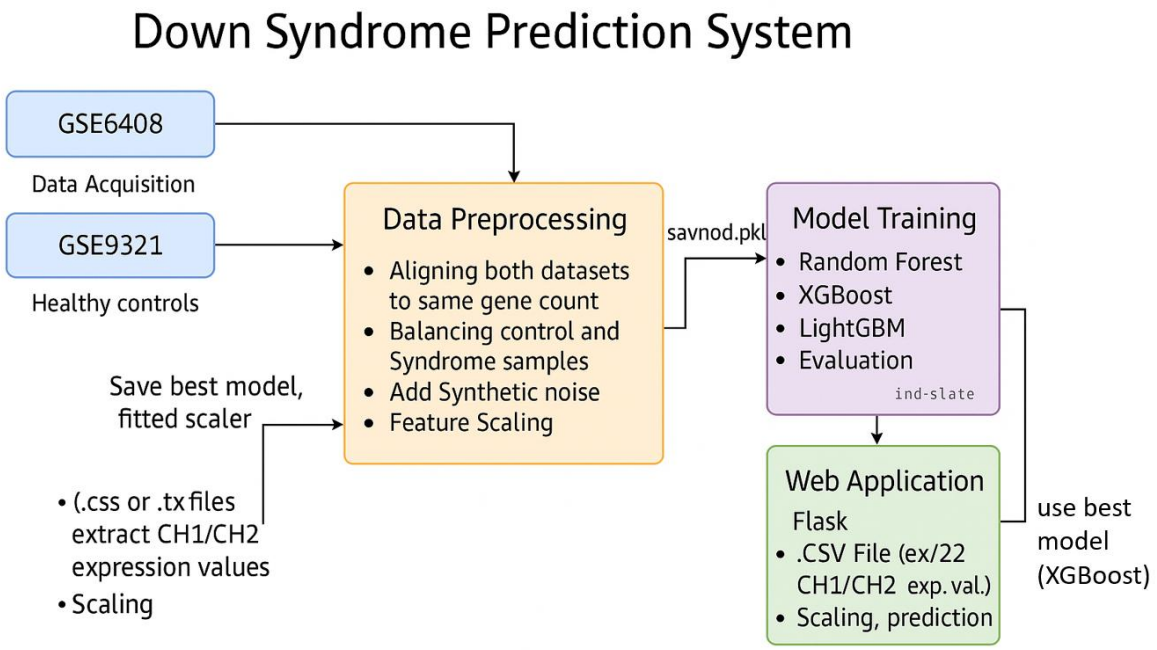

Fig. 3.2.1 Architecture of **Down Syndrome Prediction System**

The architecture of the Down Syndrome Prediction System as shown in Fig 3.2.1 is structured as a streamlined pipeline that integrates gene expression analysis with advanced machine learning techniques to accurately detect Down Syndrome from microarray data. It begins with the data acquisition phase, where two public datasets, GSE6408 and GSE9321, are collected. GSE6408 contains samples from individuals with Down Syndrome, while GSE9321 provides healthy control samples. These datasets typically come in .css, .tx, or similar file formats and contain gene expression data captured from two fluorescent channels—CH1 and CH2.

Once acquired, the data is passed into the preprocessing module, which performs several critical tasks to prepare the information for model training. Firstly, it aligns both datasets to ensure they have a consistent number of genes. This is essential because downstream analysis requires uniformity in input dimensions. The module then balances the dataset by ensuring an equal number of control and syndrome samples, a crucial step to prevent model bias. To improve model robustness and simulate real-world variability, synthetic noise is added. After that, feature scaling is applied to normalize the data, bringing all gene expression values into a standard range to improve the performance of machine learning algorithms.

The preprocessed and scaled data is then used in the model training phase, where multiple machine learning models—including Random Forest, XGBoost, and LightGBM—are trained and evaluated. These models are capable of learning complex, non-linear relationships between gene expression patterns and the presence of Down Syndrome. After evaluating performance, the best-performing model—often XGBoost—is selected, and its trained version along with the fitted scaler is saved (e.g., as savnod.pkl).

This trained model is then deployed into a Flask-based web application that serves as the user interface for diagnosis. Users can upload a .csv file containing gene expression values from new samples (such as CH1/CH2 values for 22 samples), and the system performs the same preprocessing steps, including scaling. The uploaded data is fed into the trained XGBoost model to predict whether the sample indicates Down Syndrome. The result is returned to the user through the web interface, making the system a complete, end-to-end automated prediction pipeline.

3.3 UML DIAGRAMS

3.3.1 USE CASE DIAGRAMS

A use case diagram is a visual representation that depicts the interactions between various actors and a system, capturing the ways in which users or external entities interact with the system to achieve specific goals. It is an essential tool in system analysis and design, often used in software engineering and healthcare analytics. In a use case diagram, **actors** are entities external to the system that interact with it, and **use cases** are specific functionalities or features provided by the system as illustrated in **Fig. 3.3.1.1**. These interactions are represented by lines connecting actors to use cases. The diagram helps to illustrate the scope and functionality of a system, providing a high-level view of how users or external entities will interact with it.

Actors:

1. **Researcher/Doctor**: The primary user of the system who uploads gene expression data, initiates model training, and interprets prediction results to support diagnosis and research.

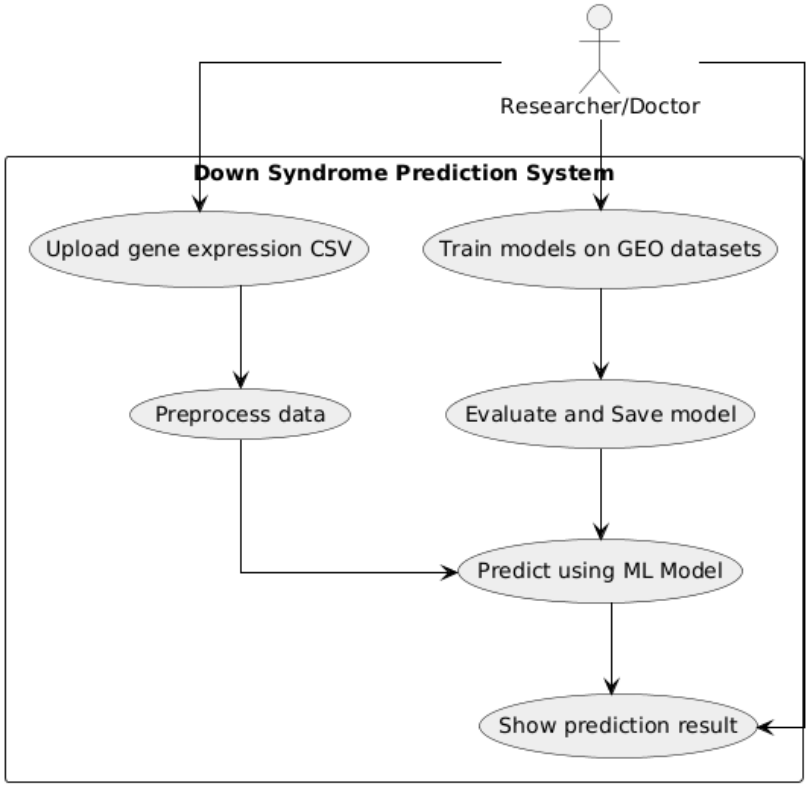

Fig. 3.3.1.1 Use Case Diagram

Use Cases:

1. **Upload Gene Expression CSV**: The researcher uploads microarray or RNA-seq gene expression data in CSV format to be used for prediction.
2. **Preprocess Data**: The system performs data preprocessing tasks such as normalization, handling missing values, noise reduction, and scaling to prepare it for prediction.
3. **Train Models on GEO Datasets**: The system uses public datasets from the Gene Expression Omnibus (GEO) to train machine learning models like Random Forest, XGBoost, and LightGBM.
4. **Evaluate and Save Model**: The trained models are evaluated for performance using accuracy, precision, recall, etc., and then saved for future predictions.
5. **Predict Using ML Model**: The system uses the pre-trained models to classify whether the uploaded sample indicates Down Syndrome or not.
6. **Show Prediction Result**: The system displays the prediction result to the Result to the researcher/doctor.

3.3.2 CLASS DIAGRAM

A class diagram is a visual representation that models the static structure of a system, showcasing the system's **classes**, their **attributes**, **methods (operations)**, and the **relationships** between them as seen in **Fig. 3.3.2.1**. It is a key tool in object-oriented design and is commonly used in software engineering to define the **blueprint** of a system. This diagram presents how various components in the Down Syndrome Prediction System interact, including data loading, preprocessing, model training, and prediction serving

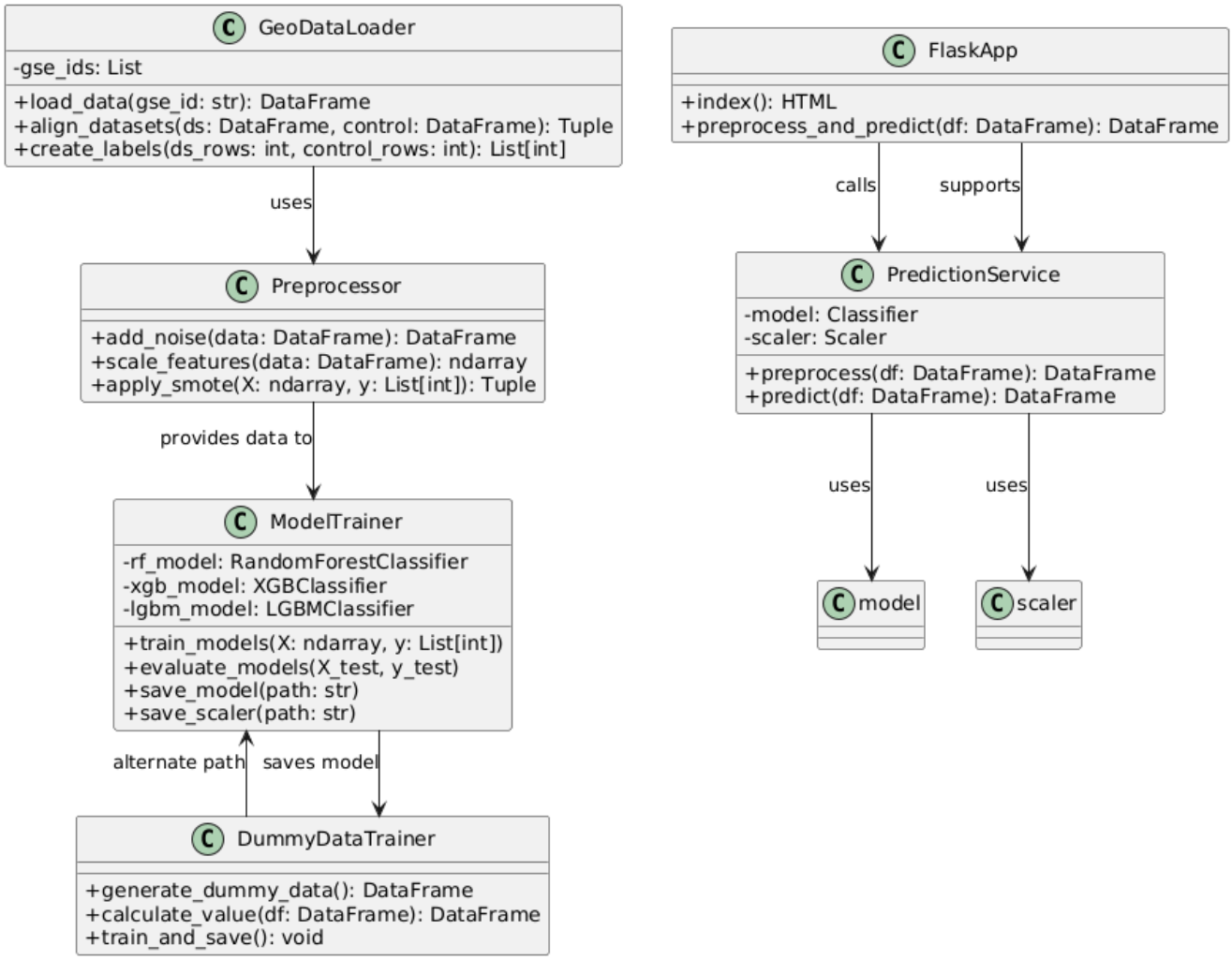

Fig. 3.3.2.1 Class Diagram

Relationships:

1. **GeoDataLoader → Preprocessor**
    - GeoDataLoader is responsible for downloading and aligning gene expression data from public repositories like GEO (Gene Expression Omnibus).
    - It uses the Preprocessor class to prepare the raw data for model training or prediction.
2. **Preprocessor → ModelTrainer**
    - The Preprocessor performs operations such as noise addition, feature scaling, and SMOTE-based balancing.
    - It provides preprocessed data to the ModelTrainer for training machine learning models.
3. **Preprocessor → ModelTrainer**
    - The Preprocessor performs operations such as noise addition, feature scaling, and SMOTE-based balancing.
    - It provides preprocessed data to the ModelTrainer for training machine learning models.
4. **ModelTrainer → DummyDataTrainer (alternate path)**
    - DummyDataTrainer is used in test environments to generate synthetic data for validation or development purposes.
    - It serves as an alternate training path to the main ModelTrainer.
5. **ModelTrainer → PredictionService (via saved model)**
    - Once models and scalers are trained, the ModelTrainer saves them for later use by the PredictionService.
6. **FlaskApp → PredictionService**
    - The FlaskApp provides the user-facing API layer. It calls the PredictionService to perform preprocessing and prediction on uploaded datasets.

7. **PredictionService → model / scaler**
    - The PredictionService uses both a trained machine learning model and a scaler to transform new input data and perform predictions.

8. **Preprocessor → ModelTrainer**
    - The Preprocessor performs operations such as noise addition, feature scaling, and SMOTE-based balancing.
    - It provides preprocessed data to the ModelTrainer for training machine learning models.
9. **ModelTrainer → DummyDataTrainer (alternate path)**
    - DummyDataTrainer is used in test environments to generate synthetic data for validation or development purposes.
    - It serves as an alternate training path to the main ModelTrainer.
10. **ModelTrainer → PredictionService (via saved model)**
    - Once models and scalers are trained, the ModelTrainer saves them for later use by the PredictionService.
11. **FlaskApp → PredictionService**
    - The FlaskApp provides the user-facing API layer. It calls the PredictionService to perform preprocessing and prediction on uploaded datasets.
12. **PredictionService → model / scaler**
    - The PredictionService uses both a trained machine learning model and a scaler to transform new input data and perform predictions.

System Flow:

1. **Data Loading:**
    - The GeoDataLoader loads the gene expression data from GEO databases and aligns disease and control datasets.
    - It also creates labels based on the dataset structure.
2. **Preprocessing:**
    - The Preprocessor adds noise, applies scaling using StandardScaler, and optionally applies SMOTE for class balance.
3. **Model Training:**
    - The ModelTrainer trains multiple models (RandomForestClassifier, XGBClassifier, and LGBMClassifier) and evaluates their performance.
    - It then saves the trained models and scalers to disk for later use.
4. **Dummy Data Handling (Alternate):**
    - In testing scenarios, DummyDataTrainer generates mock data and trains models for non-production testing purposes.
5. **Prediction:**
    - The FlaskApp receives uploaded user data (CSV), forwards it to the PredictionService, which performs preprocessing and uses the appropriate model and scaler to generate predictions.
6. **Result Delivery:**
    - The final prediction result is returned by the API to the user, helping researchers/doctors interpret whether the sample indicates Down Syndrome

3.3.3 ACTIVITY DIAGRAM FOR PREDICTING DOWN SYNDROME

An **Activity Diagram** is a type of behavioral diagram used in **UML** to illustrate the dynamic aspects of the system. It shows the **workflow of activities**, decisions, and actions, providing a clear picture of how a system behaves during a process. As shown in **Fig. 3.3.3.1**, the diagram represents the complete flow of predicting Down Syndrome based on uploaded gene expression data from CSV files.

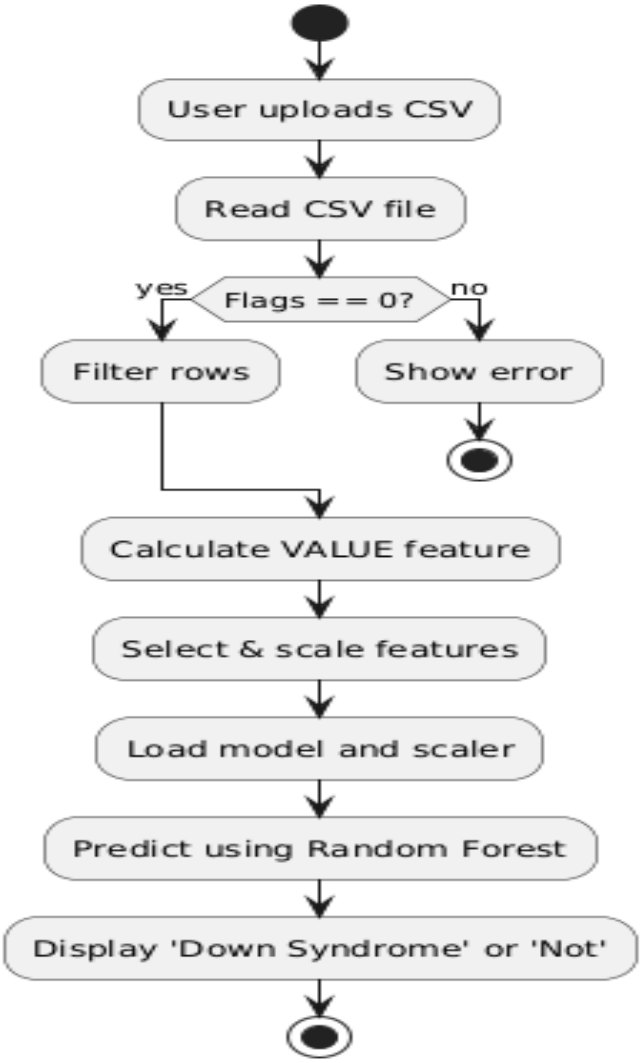

Fig. 3.3.3.1 Activity Diagram

Flow Explanation:

1. **Start:**
   The process begins when a user initiates the application.

2. **User uploads CSV file:**

   The user uploads the required CSV file of the gene expression.

3. **Read CSV file:**
   The application reads the uploaded file to parse the data for further processing.

4. **Check Flags == 0:**
   Each row in the dataset is checked for the Flags column to ensure the quality of the data.

   - If Flags == 0, the row is considered a valid good spot.

   - If Flags != 0, the system shows an error and terminates the workflow for that input.

5. **Filter rows:**
   Only valid rows with Flags == 0 are retained for further processing.

6. **Calculate VALUE feature:**
   The system computes the VALUE feature (log-ratio or other relevant expression metrics) from the signal intensities in the dataset.

7. **Select & scale features:**
   Selected features are standardized using the same scaler used during training to maintain consistency.

8. **Load model and scaler:**
   The pre-trained **XGBoost model** and **scaler** are loaded from the saved files.

9. **Predict using XGBoost:**
   The cleaned and scaled dataset is passed to the model for prediction.

10. **Display result:**
    Based on the prediction output, the system displays whether the sample indicates **'Down Syndrome'** or **'Not'**.

11. **End of workflow.**

3.3.4 SEQUENCE DIAGRAM

A sequence diagram illustrates the flow of interactions between actors and system components over time as seen in **Fig. 3.3.4.1**, emphasizing the order in which messages are exchanged to achieve specific functionalities. Actors represent external entities that interact with the system, while lifelines depict the system components involved in the process. Messages are shown as arrows, indicating the flow of information or actions between these elements. By providing a step-by-step view of workflows, sequence diagrams help in understanding and designing the dynamic behaviour of a system.

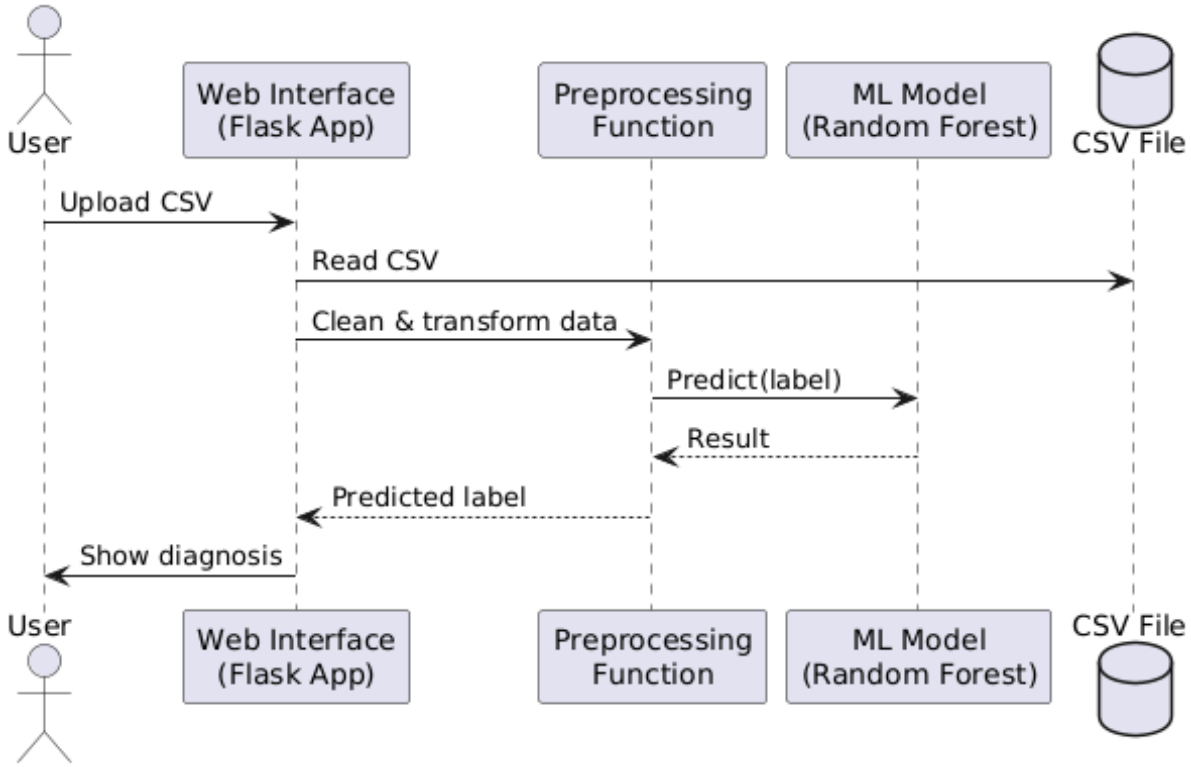

Fig. 3.3.4.1 Sequence Diagram

Key Interactions and Relationships (Specific to Down Syndrome Detection)

• User and System:

- **Upload CSV:** The user uploads a CSV file that contains microarray gene expression data from biological samples (e.g., patients or controls).
- **Show Diagnosis:** After processing and prediction, the system returns the diagnosis—whether a sample is likely associated with Down Syndrome or not.

• Web Interface (Flask App):

- **Read CSV:** The Flask application reads the uploaded CSV file. This file may contain thousands of gene expression values per sample.
- **Pass to Preprocessing:** The data is sent to a preprocessing function that handles raw biological data formatting.

• Preprocessing Function:

- **Clean & Transform Data:** This function performs multiple critical steps:
    - **Flag Filtering:** Removes bad or unreliable gene signals based on the Flags column.
    - **Background Correction:** Adjusts for background noise in fluorescence signals (e.g., CH1_SIG_Median - CH1_BKD_Median).
    - **Log Transformation / Normalization:** Ensures the gene expression values are on a comparable scale.
    - **Missing Value Imputation:** Fills in missing entries using statistical techniques.
    - **Noise Injection (Optional):** Adds controlled noise for generalization in training data.
    - **Scaling:** Applies a pre-fitted scaler (like StandardScaler) to bring data into the expected range for the ML model.
    - **Feature Selection (if applicable):** Selects only the most relevant genes used in the trained model

    .

• **ML Model (Random Forest):**

- **Predict Label:** The pre-trained Random Forest classifier takes the cleaned and scaled data and makes a prediction for each sample. The output is typically a binary label:
  - **0** – No Down Syndrome detected
  - **1** – Down Syndrome likely present
- **Return Result:** These labels are returned to the Flask interface for display.

• **System and User:**

- **Display Prediction:** The predicted labels are displayed in a user-friendly format on the web interface, possibly with visual indicators such as green (normal) or red (Down Syndrome detected).
-

3.3.5 COMPONENT DIAGRAM

A Component Diagram is a type of structural diagram used in software engineering to represent the components of a system and how they interact or depend on each other. It shows how the components (which could be software modules, subsystems, or other significant parts) are organized and connected within a system. In this diagram, each component encapsulates a set of related functionalities and interfaces as shown in **Fig. 3.3.5.1**

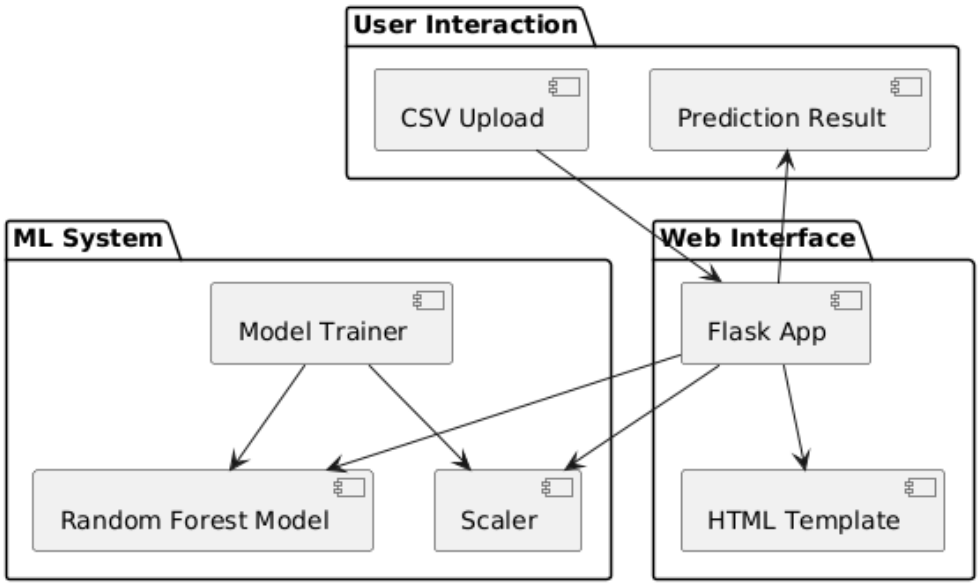

Fig. 3.3.5.1 Component Diagram

Main Components:

1. User Interaction

- **Description:**
  This component represents the front-end layer where users interact with the system. It allows users to upload gene expression data in CSV format and view the diagnosis results.
    - **CSV Upload:** Users upload microarray data files for Down Syndrome prediction.
    - **Prediction Result:** The system displays the outcome after processing and prediction.

2. Web Interface

- **Description:**
  This component acts as a bridge between the user and the machine learning backend. It is implemented using a **Flask App** and handles file uploads, communication with the ML system, and rendering results using an HTML template.
    - **Flask App:** Handles routing, manages requests, loads models and scalers, and connects components.
    - **HTML Template:** Displays the UI and diagnostic results for the user after predictions are made.

3. ML System

- **Description:**
  This is the core prediction engine responsible for training and deploying the machine learning model for Down Syndrome classification.
    - **Model Trainer:** Trains the Random Forest classifier using labeled microarray gene expression data. Also fits the scaler used for standardization.

- **Random Forest Model:** A trained model that receives preprocessed input and outputs binary predictions indicating the presence or absence of Down Syndrome.
- **Scaler:** Pre-fitted data transformer (e.g., StandardScaler) that ensures input features are on a uniform scale before being fed into the model.

3.3.6 DEPLOYMENT DIAGRAM

The Deployment Diagram illustrates the physical deployment of software components across hardware nodes in the Down Syndrome Prediction System. It captures the interactions between the user device, web server, and local model files, highlighting how the web interface, prediction logic, and machine learning components are organized and communicate in a real-world deployment scenario.

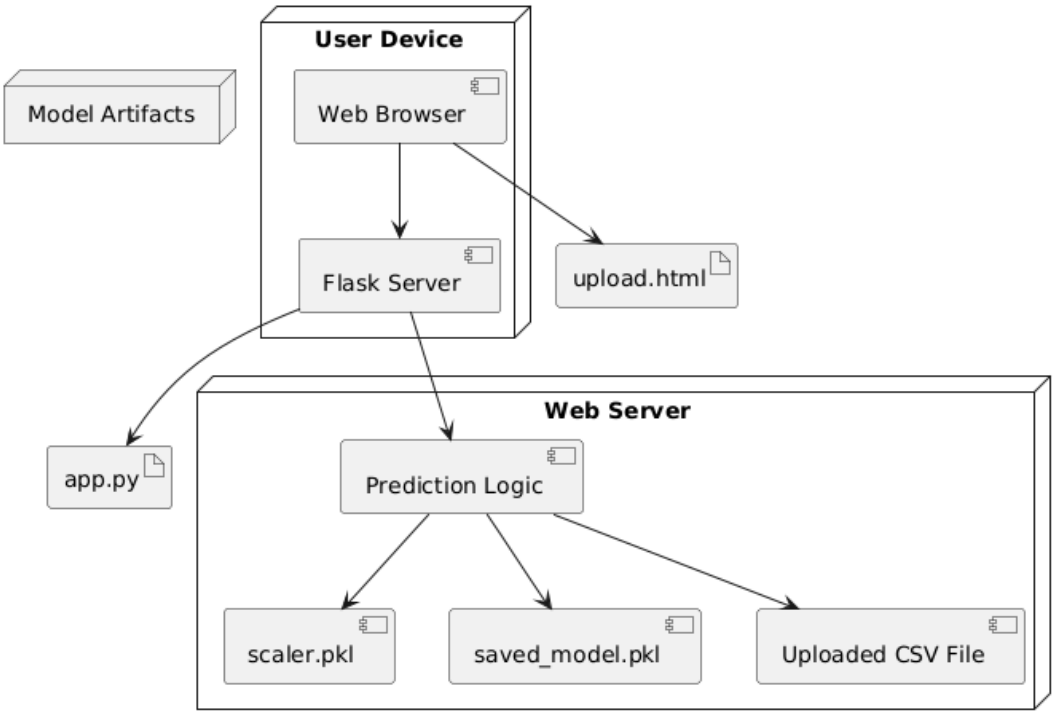

Fig. 3.3.6.1 Deployment Diagram

The deployment diagram shows how the system components are distributed across three main nodes:

- User Device: Hosts the web browser where users upload gene expression CSV files for testing. It communicates with the server using HTTP requests and displays the prediction result.
- Web Server: Runs a Flask-based application that manages user file uploads, performs preprocessing, and uses pre-trained models to classify samples as "Down Syndrome Detected" or "No Down Syndrome Detected." The server loads the serialized ML model and scaler from disk and processes the data accordingly.
- Model Artifacts (File System): Includes the saved machine learning model (saved_model.pkl) and the data scaler (scaler.pkl). These files are loaded by the Flask server during inference to ensure consistent prediction and scaling logic.

This deployment ensures seamless integration between the user interface, backend logic, and model artifacts, allowing for accurate and responsive prediction of Down Syndrome from uploaded gene expression data.

3.4 METHODOLOGY

Data Acquisition

•**Dataset Source**:

The system uses publicly available microarray gene expression datasets from the **Gene Expression Omnibus (GEO)** database, particularly datasets such as **GSE6408** and **GSE9321**. These datasets contain high-dimensional gene expression data from both Down Syndrome (DS) and control (non-DS) samples.

• **Data Format**:

Each sample is provided in CSV format with features including CH1_SIG_Median, CH1_BKD_Median, CH2_SIG_Median, CH2_BKD_Median, Flags, and normalized expression values.

• **Purpose**:

The data is used to train machine learning models that can accurately classify whether a given sample corresponds to a Down Syndrome patient or a healthy individual.

Model Steps

• **Data Preprocessing**:

- **Flag Filtering**: Only probes with Flags == 0 are considered "good quality" and retained.
- **Signal Correction**: For each channel, corrected signal values are computed by subtracting background from signal intensity (e.g., CH1_SIG_Median - CH1_BKD_Median).
- **Normalization**: Gene expression features are standardized using a **StandardScaler** to ensure all values are on the same scale.
- **Missing Values**: Any missing data is handled via imputation or row-wise deletion.
- **Balancing the Dataset**: The Synthetic Minority Over-sampling Technique (**SMOTE**) is used to address class imbalance, especially due to the smaller number of Down Syndrome samples.

Models Used

1. Random Forest Classifier

•**Description**:

Random Forest is an ensemble learning method based on decision trees. It operates by constructing a multitude of decision trees during training and outputting the class that is the mode of the classes predicted by individual trees.

• **How it Works**:

The model is trained on preprocessed and scaled gene expression data. Each tree makes an individual prediction about whether a sample indicates Down Syndrome. The forest aggregates these votes for the final classification.

• **Why it's Used**:

Random Forest is highly robust to noise and overfitting, especially suitable for high-dimensional biological data. It also provides feature importance, useful for identifying key genes involved in DS detection.

2. XGBoost Classifier

•**Description**:

XGBoost (Extreme Gradient Boosting) is a powerful, scalable implementation of gradient boosting frameworks. It builds models sequentially, optimizing errors from previous models.

•**How it Works**:

Each weak learner (a decision tree) is trained to correct the mistakes made by the previous ensemble. The model uses gradient descent on the loss function to optimize.

•**Why it's Used**:

XGBoost is known for its speed, accuracy, and ability to handle sparse or missing data, making it suitable for medical datasets where quality and consistency vary.

3. LightGBM Classifier

•**Description**:

LightGBM is a gradient boosting framework that uses tree-based learning algorithms and is optimized for speed and efficiency.

• **How it Works**:

It builds trees leaf-wise (as opposed to level-wise) and focuses on areas of the data that reduce loss the most, resulting in faster and more accurate training on large datasets.

•**Why it's Used**:

LightGBM is extremely efficient on high-dimensional data and supports parallel learning. It performs well on small-to-medium biomedical datasets and can be deployed quickly.

Model Selection and Evaluation

•**Cross-Validation**:
K-Fold Cross-Validation is used to estimate model performance and avoid overfitting.

•**Metrics**:
Models are evaluated using metrics such as:

- **Accuracy**
- **Precision**
- **Recall**
- **F1-Score**
- **ROC-AUC Score**

•Best Model:

In practical deployment, the **XGBoost model** is used due to its high interpretability, robustness, and consistently strong classification performance.

4. CODE AND IMPLEMENTATION

4.1 CODE

Iomp3.py:

```
import GEOparse
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import os
import joblib

# Step 1: Download and Parse GSE6408 Dataset (Down Syndrome)
print("Downloading GSE6408 dataset (Down Syndrome)...")
gse6408 = GEOparse.get_GEO("GSE6408", destdir="./")
print("Parsing GSE6408 expression data...")
expression_data_ds = {
    gsm_id: gsm_obj.table["VALUE"].values
    for gsm_id, gsm_obj in gse6408.gsms.items()
}
expression_data_ds = pd.DataFrame(expression_data_ds)

# Step 2: Download and Parse GSE9321 Dataset (Healthy Controls)
```

```
print("Downloading GSE9321 dataset (Healthy Controls)...")
gse9321 = GEOparse.get_GEO("GSE9321", destdir="./")
print("Parsing GSE9321 expression data...")
expression_data_control = {
    gsm_id: gsm_obj.table["VALUE"].values
    for gsm_id, gsm_obj in gse9321.gsms.items()
}
expression_data_control = pd.DataFrame(expression_data_control)

# Step 3: Align gene counts
print("Aligning datasets...")
min_genes = min(expression_data_ds.shape[0], expression_data_control.shape[0])
expression_data_ds = expression_data_ds.iloc[:min_genes, :]
expression_data_control = expression_data_control.iloc[:min_genes, :]

# Step 4: Balance sample counts
print("Balancing number of samples...")
n_ds_samples = expression_data_ds.shape[1]
expression_data_control = expression_data_control.sample(n=n_ds_samples,
random_state=42, replace=True)
expression_data_control = expression_data_control.reset_index(drop=True)
expression_data_ds = expression_data_ds.reset_index(drop=True)

# Step 5: Merge and label
print("Merging datasets and creating labels...")
merged_expression_data = pd.concat([expression_data_ds, expression_data_control], axis=0)
labels_ds = np.array([1] * expression_data_ds.shape[0])
labels_control = np.array([0] * expression_data_control.shape[0])
merged_labels = np.concatenate([labels_ds, labels_control])

# Step 6: Clean data
print("Cleaning data...")
merged_expression_data = merged_expression_data.apply(pd.to_numeric, errors='coerce')
```

```
merged_expression_data = merged_expression_data.fillna(merged_expression_data.mean())

# Step 7: Add synthetic noise
print("Adding synthetic noise to data...")
np.random.seed(42)
noise = np.random.normal(0, 0.1, merged_expression_data.shape)
synthetic_data = merged_expression_data + noise

# Step 8: Scale features
print("Scaling features...")
scaler = StandardScaler()
X_scaled = scaler.fit_transform(synthetic_data)
y = merged_labels

# Step 9: Split
print("Splitting dataset...")
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, stratify=y, random_state=42)

# Step 10: SMOTE
print("Applying SMOTE...")
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# ---------------------- Train Models ----------------------
print("\nTraining Random Forest...")
rf_model = RandomForestClassifier(random_state=42, n_jobs=-1)
rf_model.fit(X_train_resampled, y_train_resampled)

print("\nTraining XGBoost...")
xgb_model = XGBClassifier(random_state=42, n_jobs=-1, use_label_encoder=False, eval_metric='logloss')
xgb_model.fit(X_train_resampled, y_train_resampled)
```

```
print("\nTraining LightGBM...")
lgbm_model = LGBMClassifier(random_state=42, n_jobs=-1)
lgbm_model.fit(X_train_resampled, y_train_resampled)

# ---------------------- Evaluation ----------------------
print("\nEvaluating Random Forest...")
rf_preds = rf_model.predict(X_test)
rf_acc = accuracy_score(y_test, rf_preds)
print("Accuracy:", rf_acc)
print(classification_report(y_test, rf_preds))

print("\nEvaluating XGBoost...")
xgb_preds = xgb_model.predict(X_test)
xgb_acc = accuracy_score(y_test, xgb_preds)
print("Accuracy:", xgb_acc)
print(classification_report(y_test, xgb_preds))

print("\nEvaluating LightGBM...")
lgbm_preds = lgbm_model.predict(X_test)
lgbm_acc = accuracy_score(y_test, lgbm_preds)
print("Accuracy:", lgbm_acc)
print(classification_report(y_test, lgbm_preds))

# ---------------------- Accuracy Comparison Bar Chart ----------------------
print("\nGenerating accuracy comparison chart...")
model_names = ['Random Forest', 'XGBoost', 'LightGBM']
accuracies = [rf_acc, xgb_acc, lgbm_acc]

plt.figure(figsize=(8, 5))
sns.barplot(x=model_names, y=accuracies, palette=['skyblue', 'lightgreen', 'orange'])

# Highlight best model
```

```
best_index = np.argmax(accuracies)
for i, acc in enumerate(accuracies):
    plt.text(i, acc + 0.01, f"{acc:.3f}", ha='center', va='bottom', fontweight='bold')
plt.title("Model Accuracy Comparison")
plt.ylim(0, 1.1)
plt.xlabel("Model")
plt.ylabel("Accuracy")
plt.tight_layout()
plt.savefig("model_accuracy_comparison.png")
plt.show()

# ---------------------- Save the Model & Scaler ----------------------
print("\nSaving XGBoost model and scaler...")
joblib.dump(rf_model, "saved_model.pkl")
joblib.dump(scaler, "scaler.pkl")
print("Files saved successfully in:", os.getcwd())
```

app.py:

```
from flask import Flask, request, render_template
import pandas as pd
import numpy as np
import joblib

app = Flask(__name__)

# Update these paths to where your saved files are located
MODEL_PATH =
r"C:\Users\91897\AppData\Local\Programs\Python\Python312\saved_model.pkl"
SCALER_PATH = r"C:\Users\91897\AppData\Local\Programs\Python\Python312\scaler.pkl"

def load_model_or_scaler(path):
```

```
    try:
        obj = joblib.load(path)
        print(f"Loaded object of type {type(obj)} from {path}")
        return obj
    except Exception as e:
        print(f"Failed to load {path}: {e}")
        raise

model = load_model_or_scaler(MODEL_PATH)
scaler = load_model_or_scaler(SCALER_PATH)

# Verify scaler object has transform method
if not hasattr(scaler, "transform"):
    raise TypeError(f"Scaler loaded is not a scaler object. Got {type(scaler)} instead.")

def preprocess_and_predict(df):
    # Filter good spots only
    df = df[df["Flags"] == 0].copy()

    # Calculate VALUE feature
    ch1 = df["CH1_SIG_Median"] - df["CH1_BKD_Median"]
    ch2 = df["CH2_SIG_Median"] - df["CH2_BKD_Median"]
    df["VALUE"] = np.log2((ch2 + 1e-5) / (ch1 + 1e-5))

    # Select features
        features = ["CH1_SIG_Median", "CH1_BKD_Median", "CH2_SIG_Median",
"CH2_BKD_Median", "VALUE"]
    X = df[features]

    # Scale features
    X_scaled = scaler.transform(X)

    # Predict labels
```

```
    preds = model.predict(X_scaled)

    df["Predicted_Label"] = preds
    return df

@app.route("/", methods=["GET", "POST"])
def index():
    result = None
    if request.method == "POST":
        uploaded_file = request.files.get("file")
        if not uploaded_file:
            result = {"error": "No file uploaded. Please upload a CSV file."}
        else:
            try:
                df = pd.read_csv(uploaded_file)
                df_pred = preprocess_and_predict(df)
                positive_count = df_pred["Predicted_Label"].sum()
               verdict = "Down Syndrome Detected" if positive_count > 0 else "No Down Syndrome
Detected"
                result = {"verdict": verdict}
            except Exception as e:
                result = {"error": f"Error processing the file: {str(e)}"}

    return render_template("index.html", result=result)

if __name__ == "__main__":
    app.run(debug=True)
```

4.2 IMPLEMENTATION

The implementation of the Down Syndrome detection system is modular and follows a structured pipeline for preprocessing, feature selection, model loading, and prediction. The directory structure and implementation steps are described below.

Directory Structure

```
├── app.py                  # Flask web application
├── requirements.txt        # Python dependencies
├── saved_model/            # Folder for model and scaler files
│   ├── saved_model.pkl     # Trained RandomForest model
│   └── scaler.pkl          # Trained feature scaler
│
├── src/                    # Source scripts for components
│   ├── geo_loader.py       # GeoDataLoader: For loading GSE datasets
│   ├── preprocessor.py     # Preprocessor: Noise, scaling, SMOTE
│   ├── trainer.py          # ModelTrainer: Training & evaluation
│   ├── predictor.py        # PredictionService: Prediction logic
│   └── dummy_trainer.py    # DummyDataTrainer: For mock training
│
├── static/                 # Static files (if UI is styled)
│   └── css/
│       └── style.css
│
├── templates/              # HTML templates (if web interface used)
│   ├── index.html          # File upload form
│   └── result.html         # Output: "Down Syndrome" or "Not"
│
└── README.md (optional)
```

Installing Required Python Packages

Open the terminal and run:

pip install -r requirements.txt

If requirements.txt is not used:

pip install flask pandas numpy scikit-learn xgboost lightgbm imbalanced-learn matplotlib seaborn

Feature Engineering and Preprocessing

1. **CSV Upload**:
   The user uploads a CSV file via the web interface or CLI.
2. **Flag Check**:
   Rows with Flags != 0 are removed to ensure only good quality microarray data is used.
3. **VALUE Calculation**:
   The VALUE feature is calculated as:
4. VALUE = log2((CH1_SIG_Median - CH1_BKD_Median) / (CH2_SIG_Median - CH2_BKD_Median))
5. **Scaling**:
   The scaler (scaler.pkl) standardizes feature ranges using StandardScaler.

Model Loading and Prediction

The saved_model.pkl is loaded and used to predict the class label:

- 1 → Down Syndrome
- 0 → Not Down Syndrome

The prediction logic is defined in predictor.py and accessed through the Flask route in app.py.

Running the Application

1. Navigate to the root directory:
2. cd DownSyndromePredictor
3. Start the Flask server:
4. python app.py
5. Open in a browser:
6. http://127.0.0.1:5000

Optional Components

- **DummyDataTrainer**:
  Used to generate synthetic gene expression data for testing the pipeline.
- **ModelTrainer**:
  Trains and evaluates models like RandomForest, XGBoost, and LightGBM, and saves them in the saved_model/ folder.

Backend Logic – Flask Endpoint Overview

- @app.route('/'): Upload interface
- @app.route('/predict', methods=['POST']): Handles prediction logic

5. TESTING

5.1 INTRODUCTION TO TESTING

Testing is an essential phase in software development that verifies the correct functionality, usability, performance, and reliability of the application under varying conditions. It ensures the final system meets the desired specifications and behaves consistently under expected and unexpected inputs. Effective testing minimizes the risk of system failures and enhances the user experience by identifying bugs or inconsistencies early in the lifecycle.

In this project, the Down Syndrome Prediction System, testing was crucial to ensure that user CSV uploads, data preprocessing, ML-based classification, and result display were all functioning accurately and reliably. The goal was to validate that each module—from reading microarray data to classifying Down Syndrome and showing predictions through the web interface—worked as expected and maintained high diagnostic accuracy for real-world applications.

The test cases were crafted to verify the major functionalities and flow of the application, including:

• Correct transition between steps such as file upload, preprocessing, prediction, and result display

• Accurate reading and validation of user-uploaded gene expression .csv files

• Reliable preprocessing (flag filtering, noise handling, SMOTE balancing, and scaling) of microarray data

• Proper integration and functioning of the Random Forest model loaded from saved_model.pkl

• Accurate prediction output indicating the presence or absence of Down Syndrome

• Flask-based web interface responding appropriately to both valid and invalid inputs

• Graceful error messages and stability under incorrect file uploads or missing data

ACCURACY METRICS

Table 5.1 provides accuracy metrics which are used to test the performance of algorithms and the one with best measures is selected. Generally high accuracy models are preferred.

Metric	Formula	Description
Accuracy	Accuracy=TP+TNTP+TN+FP+FN\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}	Ratio of correctly predicted samples (both positives and negatives) to the total samples.
Precision	Precision=TPTP+FP\text{Precision} = \frac{TP}{TP + FP}	Of all samples predicted as positive, how many are truly positive.
Recall	Recall=TPTP+FN\text{Recall} = \frac{TP}{TP + FN}	Of all actual positives, how many were correctly identified.
F1 Score	F1 Score=2×Precision×RecallPrecision+Recall\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}	Harmonic mean of Precision and Recall, balances both.
AUC-ROC	Area under the **Receiver Operating Characteristic (ROC)** curve	Measures the ability of the model to distinguish between classes (1 = perfect, 0.5 = random).

Table 5.1 Accuracy Metrics used in project.

5.2 KEY TESTING AREAS

TC ID	Module	Test Description	Expected Outcome	Status
TC_01	Upload Page	Access root URL ("/")	File upload page loads with proper UI	Pass
TC_02	Upload Handling	Upload valid CSV file with required columns	Prediction verdict ("Down Syndrome Detected" or not)	Pass
TC_03	Empty Upload	Submit form without selecting file	Error message shown: "No file uploaded."	Pass
TC_04	Invalid Format	Upload non-CSV file (e.g., .txt or .jpg)	Error shown: "Error processing the file"	Pass
TC_05	Flag Filtering	Upload file with mixed Flags (0, -50, -100)	Only rows with Flags = 0 are processed	Pass
TC_06	VALUE Calculation	Check if VALUE is calculated from CH1/CH2 channels	VALUE column added with log2 ratios	Pass
TC_07	Scaling	Input raw values into scaler	Scaled features passed to model without error	Pass
TC_08	Prediction Logic	Upload file with Down Syndrome pattern	At least one row classified as Positive (1)	Pass
TC_09	Output Display	Check final verdict based on predictions	Verdict displayed clearly on web page	Pass

Table 5.2 Test Cases of ML based Down Syndrome Classifier

6. RESULTS

Fig 6.1 Displays the accuracy of three machine learning algorithms used.

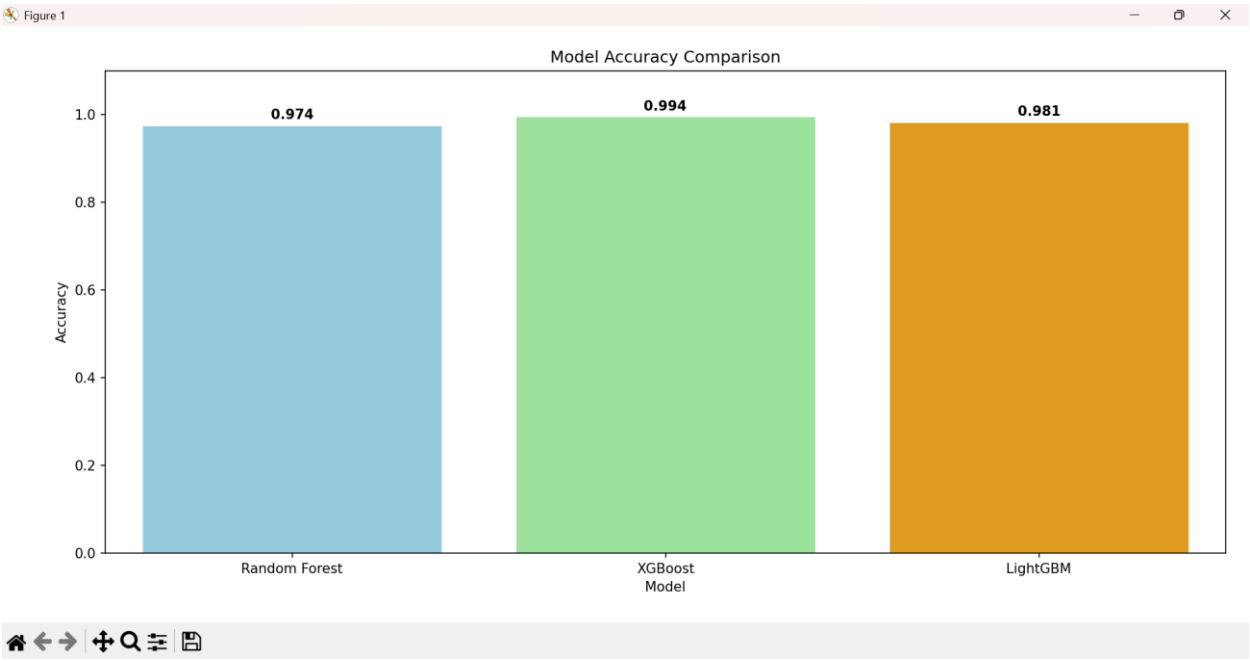

Fig 6.1 Accuracy bar chart of used machine learning algorithm

Fig 6.2 Displays the confusion matrix of Random Forest Algorithm

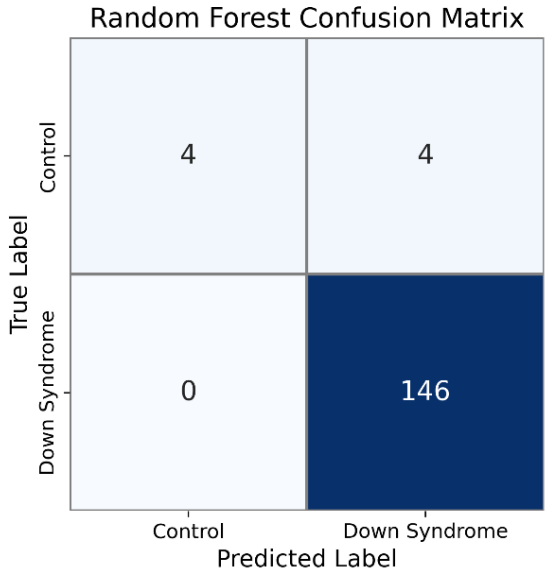

Fig 6.2 Confusion Matrix for Random Forest Algorithm

Fig 6.3 Displays the confusion matrix for XGBoost algorithm

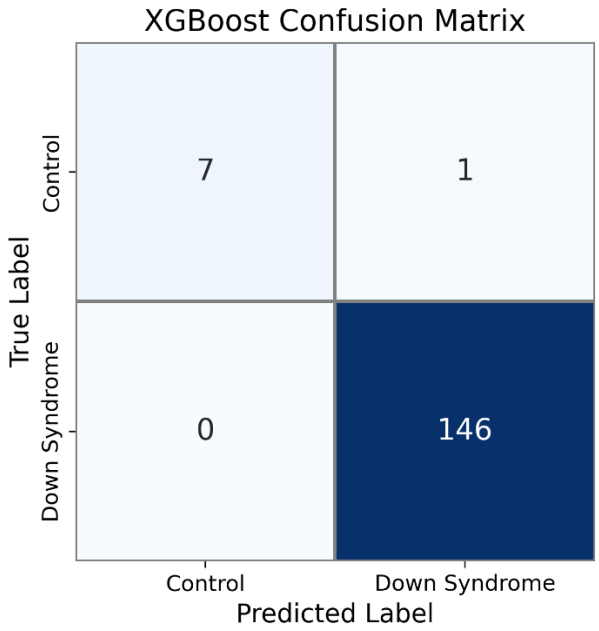

Fig 6.3 Confusion Matrix for XGBoost Algorithm

Fig 6.4 Displays the confusion matrix for LightGBM algorithm

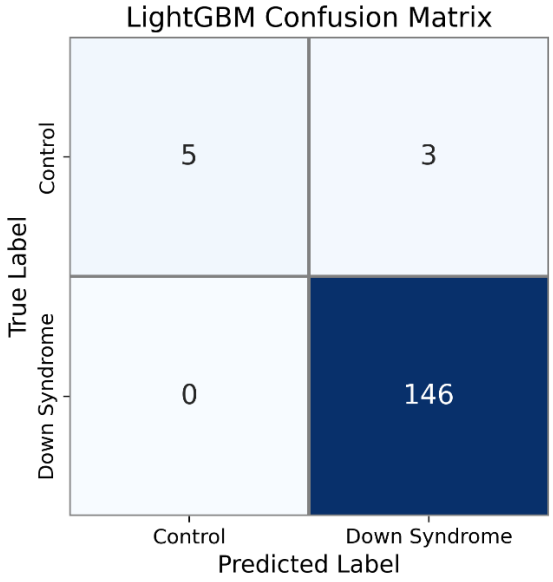

Fig 6.4 Confusion Matrix for LightGBM Algorithm

Fig 6.5 Displays the various accuracy metrics of the ML algorithms

Model	Accuracy	Precision	Recall	F1 Score	AUC-ROC
Random Forest	0.974026	0.973333	1.0	0.986486	1.0
XGBoost	0.993506	0.993197	1.0	0.996587	0.997432
LightGBM	0.980519	0.979866	1.0	0.989831	0.994863

Fig 6.5 Accuracy metrics for three algorithms

Fig. 6.6 displays the home page of the Down Syndrome Prediction system. The interface allows users to upload a CSV file containing test samples. Upon clicking the “Upload & Predict” button, the system processes the data using pre-trained machine learning models to generate diagnostic results.

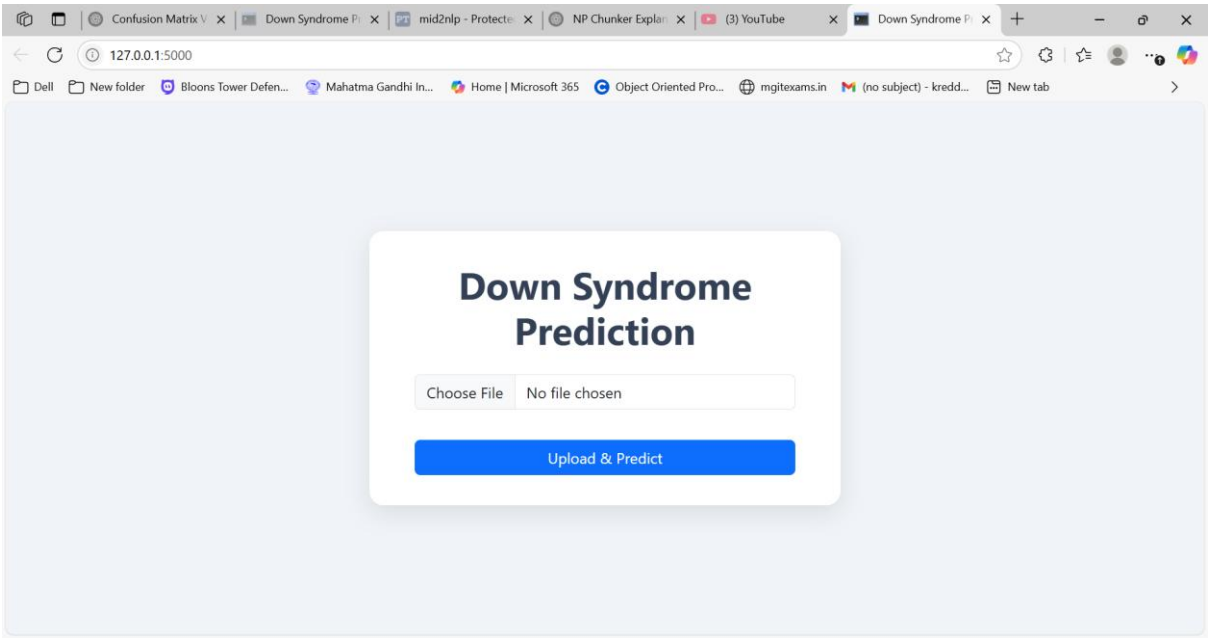

Fig. 6.6 Home Page of the Prediction System

Fig. 6.7 presents the system response when at least one sample is detected with Down Syndrome. The result section appears in a green alert box titled “Down Syndrome Detected,” followed by information on the total number of samples processed and how many were predicted positive. This helps the user immediately understand the detection outcome.

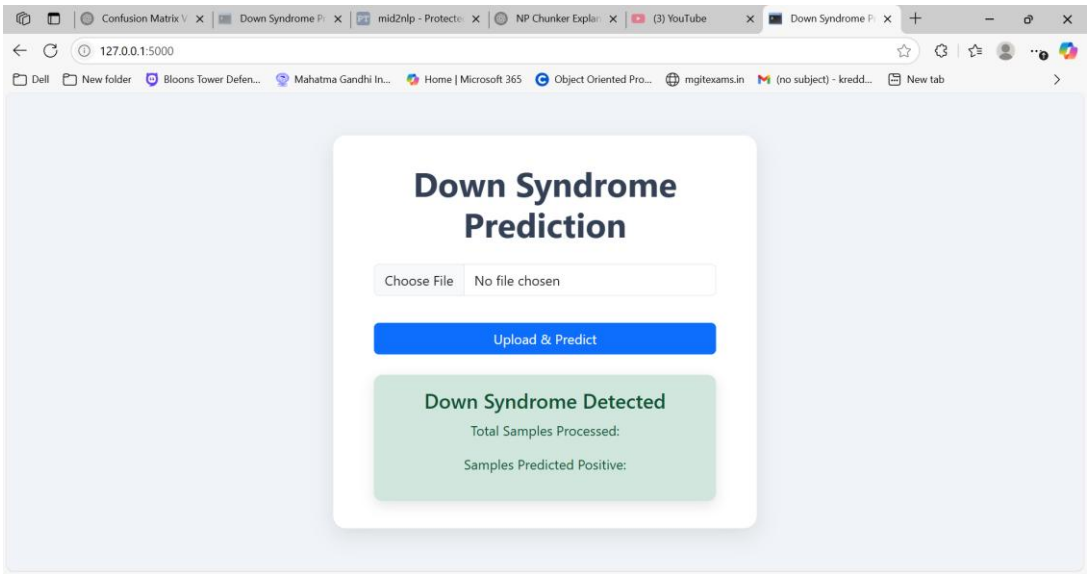

Fig. 6.7 Positive Prediction Output

Fig. 6.8 illustrates the system output when no Down Syndrome is detected in the uploaded data. The alert box labeled “No Down Syndrome Detected” informs the user that all samples were classified as negative. It includes the total number of processed samples and the count of positive predictions, which in this case would be zero.

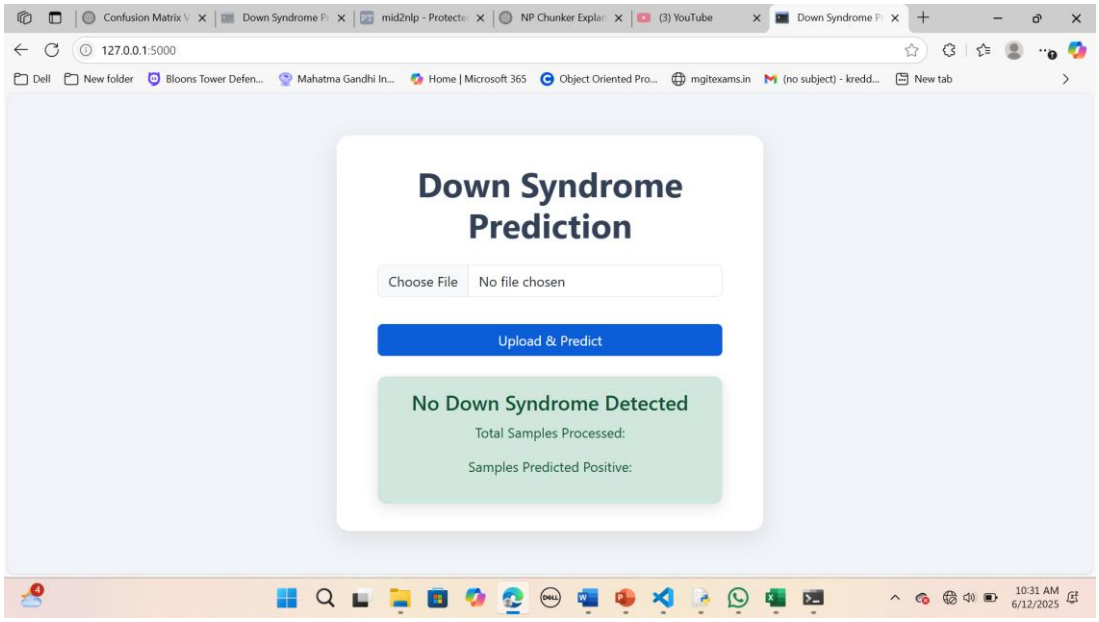

Fig 6.8 Negative Prediction Output

7. CONCLUSION AND FUTURE ENHANCEMENTS

7.1 CONCLUSION

The Down Syndrome Prediction System effectively demonstrates the integration of gene expression analysis with machine learning to assist in early diagnosis and screening. By leveraging high-dimensional microarray datasets and applying preprocessing steps such as noise handling, SMOTE balancing, and feature scaling, the system ensures clean and relevant input for the classification models.

The system incorporates a web-based interface built using Flask, where users can easily upload .csv files containing microarray data. The backend processes this data and uses a pre-trained Random Forest model to accurately classify whether a sample is likely to indicate Down Syndrome. This approach promotes non-invasive and data-driven healthcare insights, supporting researchers and healthcare providers in genetic disorder identification.

Key strengths of the system include its automation, robust backend processing, high prediction accuracy, and user-friendly design. The use of real patient data and standard bioinformatics practices like quality filtering (based on flag values) makes it a reliable tool for practical applications. This project showcases how bioinformatics, machine learning, and web technologies can be combined to address critical needs in the medical domain.

7.2 FUTURE ENHANCEMENTS

While the current version of the Down Syndrome Prediction System performs well, several improvements can be made to enhance its capability, usability, and impact:

• **Real-Time API Integration:** Enable integration with NCBI GEO or other genomic databases to directly fetch and preprocess expression datasets online.
• **Support for Multiple Genetic Disorders:** Extend the system to screen for other chromosomal or gene-expression-based disorders (e.g., Turner syndrome, Klinefelter syndrome).
• **Explainable AI (XAI):** Incorporate model explainability tools such as SHAP or LIME to help users and clinicians understand the key genes contributing to predictions.
• **Mobile/Web Dashboard:** Develop a full-featured dashboard or mobile application to visualize results and track uploaded samples, including batch processing.
• **Multi-Model Support:** Include options for users to select different ML models (e.g., XGBoost, LightGBM, or ensemble Voting Regressors) for better control and experimentation.
• **Improved Preprocessing Automation:** Automate more advanced gene filtering, normalization, and signal correction steps using established bioinformatics pipelines.
• **Data Privacy and Encryption:** Implement secure file handling and user authentication to ensure sensitive genetic data is protected during upload and processing.

These enhancements can make the system more scalable, reliable, and suited for both academic research and clinical diagnostics.

REFERENCES

[1] E. Çelik, H. O. İlhan, and A. Elbir, “Detection and estimation of Down syndrome genes by machine learning techniques,” Proceedings of the 25th Signal Processing and Communications Applications Conference (SIU), Antalya, Turkey, 2017, pp. 1–4. doi: 10.1109/SIU.2017.7960496

[2] L. Li, W. Liu, H. Zhang, Y. Jiang, X. Hu, and R. Liu, “Down syndrome prediction using a cascaded machine learning framework designed for imbalanced and feature-correlated data,” IEEE Access, vol. 7, pp. 97582–97593, 2019. doi: 10.1109/ACCESS.2019.2929681

[3] S. Ramanathan, M. Sangeetha, S. Talwai, and S. Natarajan, “Probabilistic determination of Down’s syndrome using machine learning techniques,” Proceedings of the 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Bangalore, India, 2018, pp. 126–132. doi: 10.1109/ICACCI.2018.8554392

[4] M. Rodrigues, J. Nunes, S. Figueiredo, A. Martins de Campos, and A. F. Geraldo, “Neuroimaging assessment in Down syndrome: a pictorial review,” Insights into Imaging.

[5] C. Gupta, P. Chandrashekar, T. Jin, C. He, S. Khullar, Q. Chang, and D. Wang, “Bringing machine learning to research on intellectual and developmental disabilities: taking inspiration from neurological diseases,” NPJ Digital Medicine.

[6] A. Raza, K. Munir, M. S. Almutairi, and R. Sehar, “Novel transfer learning based deep features for diagnosis of Down syndrome in children using facial images,” IEEE Access.

[7] F. L. Cibrian, Y. Chen, K. Anderson, C. M. Abrahamsson, and V. G. Motti, “Limitations in speech recognition for young adults with Down syndrome,” Universal Access in the Information Society.