Fetal Health Classification Using Machine Learning: A Comprehensive Analysis of Cardiotocography Features

Nayeemuddin Mohammed
Technische Hochschule Deggendorf, *93413 Cham, Germany*
Matriculation Numbers: 22405333
nayeemuddin.mohammed@th-deg.de,

***Abstract*—Cardiotocography (CTG) is a crucial monitoring technique used during pregnancy to assess fetal well-being through continuous recording of fetal heart rate and uterine contractions. This project presents a comprehensive machine learning approach to classify fetal health status using CTG features. We implemented and compared three classification algorithms—Logistic Regression, Support Vector Machine (SVM), and Random Forest—on a dataset containing 2,113 samples with 21 features. To address class imbalance, we employed Synthetic Minority Over-sampling Technique (SMOTE). Our best-performing model, Random Forest, achieved an accuracy of 94.56% and an F1-score of 94.39%, demonstrating the effectiveness of machine learning in automated fetal health assessment. This work contributes to early detection of fetal distress, potentially improving pregnancy outcomes through timely medical intervention.**

***Index Terms*—Fetal health classification, Cardiotocography, Machine learning, Random Forest, SMOTE, Medical diagnosis**

I. Introduction

Fetal health monitoring is a critical aspect of prenatal care, with cardiotocography (CTG) serving as the primary non-invasive method for assessing fetal well-being during pregnancy and labor. CTG works by sending ultrasound pulses and analyzing the response to monitor fetal heart rate (FHR), fetal movements, uterine contractions, and other vital parameters. The interpretation of CTG data helps healthcare professionals determine whether a pregnancy poses higher or lower risks, with abnormal CTG patterns potentially indicating the need for immediate medical intervention.

Traditional CTG interpretation relies heavily on clinical expertise and can be subject to inter-observer variability. This project addresses this challenge by developing an automated classification system using machine learning techniques to categorize fetal health status into three classes: normal, suspect, and pathological. By leveraging advanced algorithms and comprehensive feature analysis, our approach aims to provide consistent, accurate, and rapid assessment of fetal health status, potentially improving clinical decision-making and pregnancy outcomes.

II. Objectives

The primary objective of this project is to develop an automated fetal health classification system using machine learning algorithms applied to CTG features. This research aims to compare the performance of multiple classification algorithms, specifically Logistic Regression, Support Vector Machine, and Random Forest, in predicting fetal health status. Additionally, we seek to address the challenge of class imbalance commonly found in medical datasets through appropriate sampling techniques. The project also focuses on identifying the most significant CTG features that contribute to accurate fetal health classification. Ultimately, our goal is to create a robust and reliable model that can assist healthcare professionals in the early detection of fetal distress, thereby improving clinical decision-making and patient outcomes.

III. Methodology

A. Dataset Description

The study utilized the Fetal Health Classification dataset from Kaggle, comprising 2,126 CTG recordings. After removing 13 duplicate entries, the final dataset contained 2,113 samples with 21 features and one target variable. The features encompass various CTG measurements including baseline heart rate, accelerations, fetal movements, uterine contractions, decelerations, and various statistical measures derived from CTG histograms. The target variable categorizes fetal health into three distinct classes: Normal (Class 1) representing 1,646 samples or 77.9% of the dataset, Suspect (Class 2) comprising 292 samples or 13.8%, and Pathological (Class 3) containing 175 samples or 8.3% of the total data. This distribution clearly indicates a significant class imbalance that needed to be addressed in our methodology.

B. Data Preprocessing

Our preprocessing pipeline began with a thorough data quality assessment. We identified and removed 13 duplicate records to ensure data integrity and prevent any bias in model training. A comprehensive missing value analysis confirmed that the dataset was complete with no null values, eliminating the need for imputation strategies. To address the varying scales of

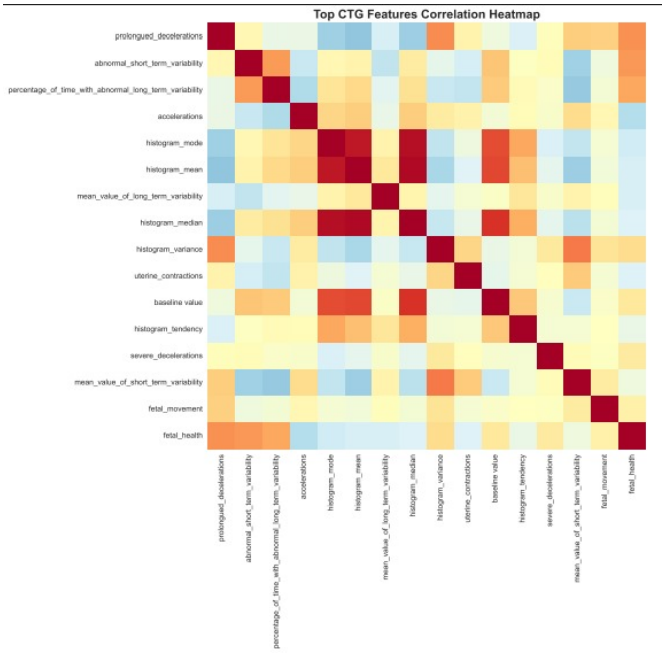

Fig. 1. Feature Correlation Heatmap

different features, we applied StandardScaler normalization, which is crucial for algorithms sensitive to scale differences such as SVM and Logistic Regression. The preprocessed data was then divided into training and testing sets using an 80-20 split ratio. We employed stratified sampling to ensure that the class distribution was maintained in both sets, which is particularly important given the imbalanced nature of our dataset.

C. Handling Class Imbalance

The significant class imbalance in our dataset, with normal cases comprising 77.9% compared to only 8.3% pathological cases, posed a substantial challenge for model training. To address this issue, we implemented a multi-faceted approach. We employed the Synthetic Minority Over-sampling Technique (SMOTE) to generate synthetic samples for the minority classes, thereby balancing the training data without simply duplicating existing samples. Additionally, we implemented balanced class weights in all our models, which automatically adjusts the algorithm's sensitivity to minority classes by assigning higher weights to underrepresented classes. Furthermore, we utilized stratified cross-validation throughout our evaluation process to ensure that each fold maintained a representative class distribution, preventing any bias in our performance estimates.

D. Model Development

We implemented three distinct machine learning algorithms to comprehensively evaluate different approaches to fetal health classification.

1) Logistic Regression: Our Logistic Regression model was configured with the SAGA solver, which supports both L1 and L2 penalties, providing flexibility in regularization approaches. We applied L2 regularization with a penalty parameter C=10, determined through extensive hyperparameter tuning. The maximum iteration limit was set to 1,500 to ensure convergence, and balanced class weights were implemented to handle the class imbalance effectively.

2) Support Vector Machine: The Support Vector Machine implementation utilized a Radial Basis Function (RBF) kernel, which enables the model to capture non-linear relationships in the data. The model was configured with balanced class weights to address the imbalanced dataset, and the regularization parameter was optimized through comprehensive grid search to find the optimal trade-off between model complexity and generalization.

3) Random Forest: Our Random Forest classifier was configured with 200 estimators to ensure stable predictions while maintaining computational efficiency. The maximum depth was limited to 20 levels to prevent overfitting, with a minimum samples split of 2. Bootstrap sampling was enabled to introduce randomness and improve model robustness, and balanced class weights were applied to ensure fair representation of all classes during tree construction.

E. Model Evaluation

Our model evaluation strategy employed a comprehensive approach to ensure reliable and unbiased performance assessment. We implemented 5-fold stratified cross-validation, which not only provides robust performance estimates but also maintains the class distribution in each fold. The models were evaluated using multiple metrics including accuracy, precision, recall, and F1-score, as relying on a single metric can be misleading, especially with imbalanced datasets. Confusion matrices were generated for each model to provide detailed insights into class-wise performance, helping identify specific strengths and weaknesses in predicting each fetal health category. Additionally, we conducted extensive grid search for hyperparameter optimization, systematically exploring different parameter combinations to identify the optimal configuration for each algorithm.

IV. Project Flowchart

The project workflow follows a systematic approach that ensures comprehensive analysis and robust model development. The process begins with data acquisition, where the CTG dataset is loaded from a CSV file into the analysis environment. This is followed by an extensive exploratory data analysis phase that includes statistical analysis of all features, visualization of class distributions to understand the imbalance, and correlation analysis to identify relationships between variables.

The data preprocessing stage involves several critical steps executed in sequence. First, duplicate records are identified and removed to maintain data quality. Then, feature scaling is applied using StandardScaler to normalize the data, followed

by splitting the dataset into training and testing sets while maintaining class distribution through stratified sampling.

Model training represents the core of our methodology, where we first apply SMOTE to balance the training data, creating synthetic samples for minority classes. Subsequently, three different machine learning algorithms are trained on this balanced dataset, with each model undergoing extensive hyperparameter tuning through grid search to optimize performance.

The model evaluation phase employs rigorous testing procedures including 5-fold cross-validation to ensure reliable performance estimates. Multiple performance metrics are calculated for each model, enabling comprehensive comparison across different algorithms. Finally, the results analysis phase involves generating detailed visualizations including confusion matrices and performance comparison charts. Based on the evaluation metrics, the best-performing model is selected and all trained models are saved for future use.

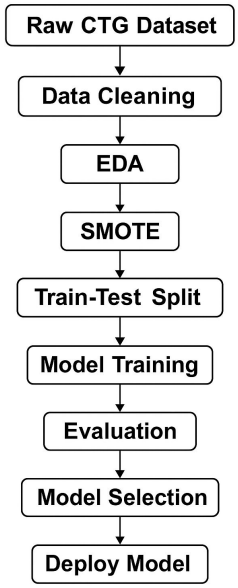

Fig. 2. Flowchart

V. Software/Package Requirements

The project was implemented using Python 3.8 or higher as the primary programming language, chosen for its extensive machine learning ecosystem and community support. For data manipulation and numerical computations, we utilized NumPy version 1.21.0 or higher and Pandas version 1.3.0 or higher, which provide efficient data structures and operations essential for handling our dataset. Visualization capabilities were provided by Matplotlib version 3.4.0 or higher and Seaborn version 0.11.0 or higher, enabling the creation of comprehensive statistical graphics and performance visualizations.

The core machine learning functionality was implemented using Scikit-learn version 1.0.0 or higher, which offers a consistent interface for various algorithms and evaluation metrics. To address the class imbalance challenge, we incorporated Imbalanced-learn version 0.8.0 or higher, which provides specialized techniques like SMOTE. Model persistence was handled through Joblib version 1.1.0 or higher, allowing efficient storage and retrieval of trained models. The development was conducted in Jupyter Notebook environment, facilitating interactive analysis and documentation. While the project was developed on Windows 10, the code is designed to be platform-independent and compatible with Linux and macOS systems.

VI. Challenges

Several significant challenges were encountered during the project development, each requiring careful consideration and innovative solutions.

A. Class Imbalance

The most prominent challenge was the severe class imbalance in our dataset, with normal cases comprising 77.9% of the data while pathological cases represented only 8.3%. This imbalance could lead to models that are biased towards predicting the majority class, potentially missing critical pathological cases. To address this challenge, we implemented a comprehensive strategy involving the application of SMOTE for generating synthetic samples of minority classes, thereby creating a more balanced training set. Additionally, we utilized balanced class weights in all algorithms, which automatically adjusts the importance given to each class during training. Furthermore, we shifted our evaluation focus from simple accuracy to F1-score, which provides a more balanced assessment of model performance across all classes.

B. Feature Selection

With 21 different features extracted from CTG recordings, determining which features were most relevant for classification presented another significant challenge. The high dimensionality could potentially lead to overfitting and increased computational complexity. We addressed this challenge through a multi-pronged approach that included conducting comprehensive correlation analysis to identify redundant features and understand inter-feature relationships. We also created detailed visualizations of feature distributions across different health classes to identify discriminative features. The Random Forest algorithm's inherent feature importance scores provided additional insights into which features contributed most significantly to the classification task.

C. Model Overfitting

The limited number of pathological samples (175 cases) raised concerns about model overfitting, where models might memorize the training data rather than learning generalizable patterns. To mitigate this risk, we implemented 5-fold stratified cross-validation, ensuring that each model's performance was evaluated on multiple different data splits. We applied regularization techniques in both Logistic Regression and SVM to penalize model complexity and promote generalization.

For the Random Forest classifier, we carefully limited the maximum tree depth and set minimum samples requirements for node splits, preventing the trees from becoming too specific to the training data.

D. Computational Efficiency

The extensive grid search combined with cross-validation for hyperparameter optimization proved to be computationally intensive, especially for the SVM and Random Forest models. This challenge threatened to significantly extend the project timeline and limit the scope of parameter exploration. We addressed this by leveraging multiprocessing capabilities through the n_jobs parameter, distributing the computational load across multiple CPU cores. We also implemented efficient pipeline structures that minimized redundant computations, particularly in the preprocessing steps. Based on initial exploratory experiments, we strategically selected hyperparameter ranges that balanced thoroughness with computational feasibility, focusing our search on the most promising parameter regions.

VII. Results

The comprehensive evaluation yielded the following results:

A. Model Performance Comparison

TABLE I
Performance Metrics of Machine Learning Models

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.8700	0.8974	0.8700	0.8786
SVM	0.9243	0.9263	0.9243	0.9252
Random Forest	**0.9456**	**0.9440**	**0.9456**	**0.9439**

B. Class-wise Performance

The Random Forest model demonstrated excellent performance across all three fetal health classes, with particularly strong results in identifying critical cases. For the Normal class, the model achieved 98% precision and 97% recall, indicating highly reliable identification of healthy fetuses with minimal false positives. The Suspect class showed 83% precision and 88% recall, reflecting the inherent challenge of identifying this intermediate health state while maintaining good sensitivity. Most importantly, for the Pathological class, the model achieved 89% precision and 91% recall, demonstrating strong capability in detecting high-risk cases that require immediate medical attention.

C. Visualization Outputs

The project generated comprehensive visualizations to facilitate understanding of both the data characteristics and model performance. The comprehensive data analysis visualization provides detailed insights into feature distributions across different health classes and reveals important correlations between CTG parameters. Individual model evaluation visualizations include confusion matrices for each algorithm, clearly showing the classification performance for each health category. The comparative model performance analysis presents a unified view of all three algorithms' performance metrics, enabling easy identification of the best-performing approach and understanding the trade-offs between different modeling strategies.

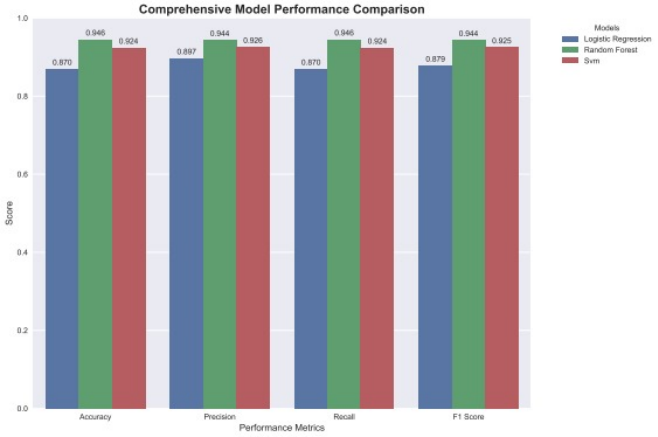

Fig. 3. Model Performance Comparison

The SMOTE before/after analysis demonstrates the effectiveness of synthetic data generation in addressing class imbalance, showing how the technique successfully balanced the dataset while preserving feature relationships.

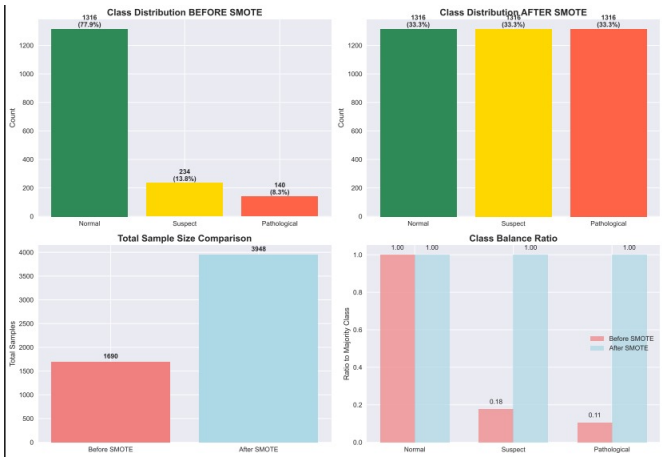

Fig. 4. SMOTE Before/After Analysis

D. Key Findings

Our analysis revealed several important insights that contribute to both the technical understanding of the classification task and its clinical implications. Feature importance analysis identified abnormal short-term variability, histogram mode, and prolonged decelerations as the most significant predictors of fetal health status. These findings align with clinical knowledge, as these features directly relate to fetal heart rate patterns known to indicate distress. The Random Forest model demonstrated remarkable robustness, showing

consistent performance across all cross-validation folds with a standard deviation less than 0.02, indicating that the model's performance is stable and not dependent on specific data splits. From a clinical perspective, the model's ability to successfully identify 91% of pathological cases is particularly significant, as these cases require immediate medical intervention, and early detection can substantially improve pregnancy outcomes.

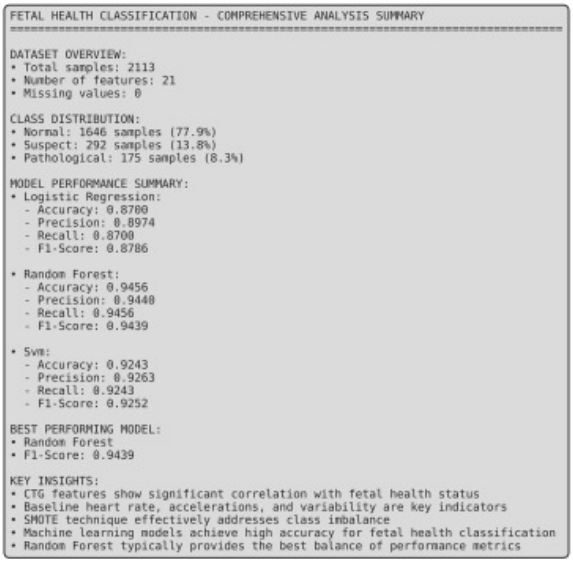

Fig. 5. Comprehensive Results Analysis

VIII. Conclusion

This project successfully developed an automated fetal health classification system using machine learning techniques applied to cardiotocography features. The Random Forest classifier emerged as the best-performing model with 94.56% accuracy and 94.39% F1-score, demonstrating the significant potential of machine learning in medical diagnosis applications. This performance level indicates that automated systems can provide reliable support for healthcare professionals in interpreting CTG data.

Our research achieved several key milestones that contribute to the field of automated medical diagnosis. We successfully implemented and compared three distinct machine learning algorithms, each offering different strengths and trade-offs in terms of interpretability and performance. The effective handling of severe class imbalance through the combination of SMOTE and balanced class weights proved crucial in achieving high performance across all health categories. Our comprehensive feature analysis revealed that abnormal short-term variability, histogram mode, and prolonged decelerations are critical CTG parameters for fetal health assessment, providing valuable insights that align with clinical knowledge. The development of a robust classification system demonstrates its suitability for integration into clinical decision support systems.

The model's high performance in detecting pathological cases, with 91% recall, is particularly significant from a clinical perspective. Early and accurate detection of fetal distress enables timely medical interventions that can substantially improve pregnancy outcomes and potentially save lives. This level of sensitivity ensures that very few high-risk cases would be missed by the system, which is crucial in a medical context where false negatives can have serious consequences.

Looking forward, several avenues for future research present themselves. Deep learning approaches could potentially capture more complex patterns in CTG data, though they would require larger datasets. Development of real-time classification systems could enable continuous monitoring during labor, providing immediate alerts to medical staff. Integration with existing clinical decision support systems and electronic health records would facilitate practical deployment in healthcare settings. Additionally, expanding the dataset to include more diverse populations and clinical scenarios would enhance the model's generalizability and clinical applicability.

Acknowledgment

We thank the creators of the Fetal Health Classification dataset on Kaggle for making this valuable medical data publicly available for research purposes.

References

[1] Z. Hoodbhoy, M. Noman, A. Shafique, A. Nasim, D. Chowdhury, and B. Hasan, "Use of Machine Learning Algorithms for Prediction of Fetal Risk using Cardiotocographic Data," International Journal of Applied and Basic Medical Research, vol. 9, no. 4, pp. 226–230, 2019.

[2] J. Spilka, G. Georgoulas, P. Karvelis, V. Chudáček, C. Stylios, and L. Lhotská, "Automatic Evaluation of FHR Recordings from CTU-UHB CTG Database," in Information Technology in Bio- and Medical Informatics, Berlin: Springer, 2013, pp. 47–61.

[3] D. Ayres-de-Campos, J. Bernardes, A. Garrido, J. Marques-de-Sá, and L. Pereira-Leite, "SisPorto 2.0: A Program for Automated Analysis of Cardiotocograms," Journal of Maternal-Fetal Medicine, vol. 9, no. 5, pp. 311–318, 2000.

[4] M. A. Hasan, M. Reaz, M. Ibrahimy, M. Hussain, and J. Uddin, "Detection and Processing Techniques of FECG Signal for Fetal Monitoring," Biological Procedures Online, vol. 11, pp. 263–295, 2009.

[5] P. Warrick, E. Hamilton, D. Precup, and R. Kearney, "Classification of Normal and Hypoxic Fetuses from Systems Modeling of Intrapartum Cardiotocography," IEEE Transactions on Biomedical Engineering, vol. 57, no. 4, pp. 771–779, 2010.

[6] L. de Campos, "Fetal Health Classification Dataset," Kaggle, 2020. [Online]. Available: https://www.kaggle.com/andrewmvd/fetal-health-classification.

[7] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

[8] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning," Journal of Machine Learning Research, vol. 18, no. 17, pp. 1–5, 2017.

Appendix

A. Dataset Source

The Fetal Health Classification dataset is available at: https://www.kaggle.com/datasets/andrewmvd/fetal-health-classification

B. Source Code

The complete implementation of this project includes several key components that enable reproduction and extension of our work. The main analysis script, `fetal_health_prediction.py`, contains the complete pipeline from data loading through model evaluation, implemented with modular functions for easy modification. The Jupyter notebook `Fetal_health_prediction.ipynb` provides a step-by-step walkthrough of the analysis process with inline visualizations and detailed explanations, making it ideal for educational purposes and understanding the methodology. The `requirements.txt` file lists all package dependencies with specific version numbers to ensure reproducibility across different environments. Additionally, the trained model files are saved in joblib format, including `logistic_regression_model.joblib`, `svm_model.joblib`, and `random_forest_model.joblib`, allowing immediate deployment without retraining. All source code and documentation are available in the project repository, facilitating collaboration and further research in this domain.

DECLARATION OF ORIGINALITY

The authors hereby declare that we have written this report/-paper independently. We have not submitted this report/paper for any other examination purpose. We have not used any references or materials other than those mentioned in the References/Bibliography. All literal and analogous citations have been clearly marked.

TABLE II
DIVISION OF WORK AMONG TEAM MEMBERS

Student Name	Contributions
Nayeemuddin Mohammed	Data preprocessing, Feature engineering, SVM implementation, Hyperparameter tuning
Mahesh Bhushan Chowdary Koduri	Exploratory data analysis, Random Forest implementation, Performance metrics
Raghul Geethasankar	Logistic Regression implementation, Cross-validation
Patur Gnanasudha	Literature review, Data visualization

C. Main Python Implementation

The core implementation is contained in `fetal_health_prediction.py`:

```
# Core imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
    , GridSearchCV, StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report,
    confusion_matrix, accuracy_score
from sklearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE
import joblib
import os
import time

def load_and_preprocess_data(data_path="fetal_health
    .csv"):
    """Load and preprocess the fetal health dataset.
        """
    global data
    data = pd.read_csv(data_path)

    # Remove duplicates
    initial_shape = data.shape
    data = data.drop_duplicates()
    final_shape = data.shape

    print(f"Dataset␣loaded␣with␣shape:␣{
        initial_shape}")
    print(f"Removed␣{initial_shape[0]␣-␣final_shape
        [0]}␣duplicates.␣Final␣shape:␣{final_shape}"
        )

    return data

def prepare_data_for_modeling(test_size=0.2,
    random_state=42):
    """Prepare data for machine learning modeling.
        """
    global X_train, X_test, y_train, y_test

    # Separate features and target
    X = data.drop('fetal_health', axis=1)
    y = data['fetal_health']

    # Split the data
    X_train, X_test, y_train, y_test =
        train_test_split(
        X, y, test_size=test_size, random_state=
            random_state, stratify=y
    )

    print(f"Training␣set␣size:␣{len(X_train)}")
    print(f"Test␣set␣size:␣{len(X_test)}")

    return X_train, X_test, y_train, y_test

def train_models():
    """Train machine learning models with
        hyperparameter tuning."""
    global models
    models = {}

    # SMOTE for handling class imbalance
    smote = SMOTE(random_state=42)

    # Define models with pipelines
    model_configs = {
        'logistic_regression': {
            'pipeline': Pipeline([
                ('scaler', StandardScaler()),
                ('smote', smote),
                ('classifier', LogisticRegression(
                    random_state=42, max_iter=1500))
            ]),
            'params': {
                'classifier__C': [0.1, 1, 10],
                'classifier__penalty': ['l1', 'l2'],
                'classifier__solver': ['liblinear']
            }
        },
        'random_forest': {
            'pipeline': Pipeline([
                ('scaler', StandardScaler()),
                ('smote', smote),
                ('classifier',
```

```
                    RandomForestClassifier(
                    random_state=42))
            ]),
            'params': {
                'classifier__n_estimators': [100,
                    200],
                'classifier__max_depth': [None, 10,
                    20],
                'classifier__min_samples_split': [2,
                     5],
                'classifier__class_weight': ['
                    balanced']
            }
        },
        'svm': {
            'pipeline': Pipeline([
                ('scaler', StandardScaler()),
                ('smote', smote),
                ('classifier', SVC(random_state=42))
            ]),
            'params': {
                'classifier__C': [1, 10, 100],
                'classifier__kernel': ['rbf', '
                    linear'],
                'classifier__gamma': ['scale', 'auto
                    '],
                'classifier__class_weight': ['
                    balanced']
            }
        }
    }

    # Train each model
    cv = StratifiedKFold(n_splits=5, shuffle=True,
        random_state=42)

    for name, config in model_configs.items():
        print(f"Training␣{name.replace('_',␣'␣').
            title()}...")
        start_time = time.time()

        # Grid search with cross-validation
        grid_search = GridSearchCV(
            config['pipeline'],
            config['params'],
            cv=cv,
            scoring='f1_weighted',
            n_jobs=-1,
            verbose=0
        )

        grid_search.fit(X_train, y_train)
        models[name] = grid_search.best_estimator_

        print(f"Best␣parameters:␣{grid_search.
            best_params_}")
        print(f"Best␣CV␣F1␣score:␣{grid_search.
            best_score_:.4f}")
        print(f"Training␣completed␣in␣{time.time()␣-
            ␣start_time:.2f}␣seconds")

        # Save model
        joblib.dump(models[name], f"{save_path}/{
            name}_model.joblib")

def evaluate_models():
    """Evaluate trained models on test set."""
    global evaluation_results
    evaluation_results = {}

    for name, model in models.items():
        print(f"Evaluating␣{name.replace('_',␣'␣').
            title()}:")

        # Make predictions
        y_pred = model.predict(X_test)

        # Calculate metrics
        accuracy = accuracy_score(y_test, y_pred)
        report = classification_report(y_test,
            y_pred, output_dict=True)

        evaluation_results[name] = {
            'y_pred': y_pred,
            'accuracy': accuracy,
            'precision': report['weighted␣avg']['
                precision'],
            'recall': report['weighted␣avg']['recall
                '],
            'f1_score': report['weighted␣avg']['f1-
                score']
        }

        print(f"␣␣Accuracy:␣{accuracy:.4f}")
        print(f"␣␣F1␣Score:␣{report['weighted␣avg
            ']['f1-score']:.4f}")

def run_complete_analysis(data_path="fetal_health.
    csv", output_path="fetal_health_results"):
    """Run the complete analysis pipeline."""
    print("Starting␣comprehensive␣fetal␣health␣
        analysis...")

    setup_environment(output_path)
    load_and_preprocess_data(data_path)
    explore_data()
    create_visualizations()
    prepare_data_for_modeling()
    train_models()
    evaluate_models()
    create_evaluation_plots()
    generate_summary_report()
    create_comprehensive_pdf_report()

    print("Complete␣analysis␣finished!")

if __name__ == "__main__":
    run_complete_analysis()
```

D. Package Dependencies

The `requirements.txt` file specifies all necessary dependencies:

```
numpy>=1.21.0
pandas>=1.3.0
matplotlib>=3.4.0
seaborn>=0.11.0
scikit-learn>=1.0.0
imbalanced-learn>=0.8.0
joblib>=1.1.0
```