Proposed Methodology/Algorithms:

The "Fetal Health Examiner" project employs a combination of preprocessing techniques and machine learning algorithms for Cardiotocogram (CTG) classification. Outline of the methodology and algorithms involved in the project are:

1) Data Preprocessing:

Data preprocessing is essential to prepare the input data for model training. This involves handling missing values and scaling features

ALGORITHM:

HANDLING MISSING VALUE

- Strategy: Impute missing values using the mean of the feature column.
- Implementation:
    - Use the SimpleImputer class from sklearn.impute module with the 'mean' strategy.
    - Fit the imputer on the training data and transform both training and testing data.

FEATURE SCALING

- Strategy: Standardize feature values to have zero mean and unit variance.
- Implementation:
    - Utilize the StandardScaler class from sklearn.preprocessing module.
    - Fit the scaler on the training data and transform both training and testing data.

```
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
# Handle missing values
imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)
```

```
# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

2) model training

2.1)Support Vector Machine(SVM)

SVM is a supervised learning algorithm used for classification tasks. Finds the optimal hyperplane that best separates the classes in the feature space.

Algorithm

from sklearn.svm import SVC

Train SVM classifier

svm_classifier = SVC(kernel='rbf', C=1.0)

svm_classifier.fit(X_train, y_train)

2.2) Logistic Regression:

Logistic Regression is a linear model used for binary classification tasks. It estimates the probability that a given instance belongs to a particular class.

from sklearn.linear_model import LogisticRegression

Train Logistic Regression classifier

logreg_classifier = LogisticRegression()

logreg_classifier.fit(X_train, y_train)

2.3) Random Forest:

Random Forest is an ensemble learning method that builds multiple decision trees during training and combines their predictions to improve accuracy and reduce overfitting.

```
from sklearn.ensemble import RandomForestClassifier

# Train Random Forest classifier

rf_classifier = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)

rf_classifier.fit(X_train, y_train)
```

3) Choosing Best Algorithm:

After training multiple models, we evaluate their performance using appropriate metrics such as accuracy, F1 score, precision, and recall. The model with the highest performance metrics is selected as the best algorithm.

algorithm

Predictions from different classifiers:

- Generate predictions for the test dataset using trained classifiers for SVM, Logistic Regression, and Random Forest algorithms.
- Store the predictions in variables: `svm_pred`, `logreg_pred`, and `rf_pred`.

Evaluate accuracy and F1 score:

- Calculate the accuracy and F1 score for each classifier based on the predicted and true labels of the test dataset.
- Use the `accuracy_score` and `f1_score` functions from the sklearn.metrics module.
- Compute accuracy and F1 score for SVM, Logistic Regression, and Random Forest classifiers.
- Store the accuracy and F1 score values in variables: `svm_accuracy`, `logreg_accuracy`, `rf_accuracy`, `svm_f1`, `logreg_f1`, and `rf_f1`.

Compare performance metrics and choose the best algorithm:

- Compare the accuracy and F1 score of SVM, Logistic Regression, and Random Forest classifiers.
- Select the classifier with the highest combined accuracy and F1 score as the best algorithm.

- Store the best algorithm and its corresponding performance metrics in the variable `best_algorithm`.

In this proposed methodology, data preprocessing ensures that the input data is properly formatted and scaled for model training. We then train multiple classification algorithms, including SVM, Logistic Regression, and Random Forest. Finally, we evaluate their performance using accuracy and F1 score metrics to select the best algorithm for CTG classification.

Description of Implementation Strategies

1)Using ML Model Algorithms from scikit-learn:

- Description: Implement machine learning algorithms from the scikit-learn library for classification tasks. This involves selecting appropriate algorithms, training them on the data, and evaluating their performance using metrics such as accuracy, precision, recall, and F1-score.
- Implementation: Utilize algorithms such as SVM, Logistic Regression, and Random Forest from the scikit-learn library for classification tasks.

2)Data Preprocessing using pandas and numpy:

- Description: Perform data preprocessing tasks such as data cleaning, normalization, and handling missing values using the pandas and numpy libraries in Python.
- Implementation:
    - Data Normalization:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)
```

- Removing Null Data:

```
import pandas as pd

# Remove rows with null values

data_cleaned = data.dropna()
```

These implementation strategies leverage popular Python libraries such as scikit-learn, pandas, and numpy to perform machine learning tasks efficiently. By utilizing pre-built functions and methods from these libraries, the implementation process is streamlined, allowing for faster development and experimentation with machine learning models.

Module Division:

1)Data Preprocessing:

- Responsible for cleaning, transforming, and preparing the raw CTG data for model training.
- Includes tasks such as handling missing values and scaling features.

2)Model Creation and Selection:

- Involves the development and training of machine learning models for CTG classification.
- Includes selecting appropriate algorithms (e.g., SVM, Logistic Regression, Random Forest) and evaluating model performance.

3)Model Evaluation:

- Focuses on evaluating the performance of trained models using metrics such as accuracy, precision, recall, and F1-score.
- Helps in comparing different models and selecting the best-performing one for deployment.

4)User Authentication:

- Manages user authentication and authorization within the system.
- Ensures secure access to the application and restricts unauthorized users from accessing sensitive data.

5)User Interface (UI):

- Deals with the design and implementation of the graphical user interface (GUI) for the application.

- Provides an interactive platform for users to upload CTG data, view analysis results, and interact with the system.

6)Graph Generation:

- Generates visualizations and graphs to represent analysis results and other relevant information.
- Enhances the interpretability of data and model outputs, aiding users in understanding and making informed decisions.

7)Integration:

- Combines the machine learning models with the user interface to create a functional application.
- Enables seamless interaction between the user interface and backend model functionalities, allowing users to perform CTG analysis effectively.

Each module plays a crucial role in the overall functionality of the "Fetal Health Examiner" project, contributing to different aspects such as data processing, model development, user interaction, and system integration.