

Fetal Distress based on Cardiotocography Data

Imaad Inayath
Student of Information Science
Engineering
Presidency University Bengaluru
Bengaluru, India
201910100844@presidencyuniversity.in

Emani Hemanth Reddy
Student of Information Science
Engineering
Presidency University Bengaluru
Bengaluru, India
201910101248@presidencyuniversity.in

Aryan Shiv R
Student of Information Science
Engineering
Presidency University Bengaluru
Bengaluru, India
201910100446@presidencyuniversity.in

Ajay Reddy N
Student of Information Science
Engineering
Presidency University Bengaluru
Bengaluru, India
201910100664@presidencyuniversity.in

Abhishek Kumar Shandilya
Student of Information Science
Engineering
Presidency University Bengaluru
Bengaluru, India
201910100546@presidencyuniversity.in

Dr. Sulaiman Syed Mohamed
Faculty of Information Science
Engineering
Presidency University Bengaluru
Bengaluru, India
sulaiman.syedmohamed@presidencyuniversity.in

Abstract— Fetal distress is a medical term used to describe a situation where the health and well-being of a fetus are compromised during pregnancy or childbirth. Several possible causes of fetal distress include lack of oxygen, infections, fetal anomalies, and placental problems. Some of the signs and symptoms of fetal distress include Abnormal heart rate patterns: The fetal heart rate may be faster or slower than usual, or there may be irregular patterns in the heart rate. Decreased fetal movement: The fetus may be less active than usual, indicating a potential problem. Meconium-stained amniotic fluid: Meconium is a dark green substance that can be present in the amniotic fluid if the fetus is experiencing distress. Abnormal levels of amniotic fluid: Too little or too much amniotic fluid can be a sign of fetal distress. If fetal distress is suspected, prompt medical intervention is necessary to prevent serious complications, including brain damage or stillbirth. Depending on the cause of fetal distress, treatment may involve increasing the oxygen supply to the fetus, performing an emergency delivery, or administering medications to the mother. Cardiotocography monitors two vital parameters, i.e. Fetal heart rate (FHR) and uterine contractions (UC). These time series data can be used to detect fetal distress.

Keywords—Fetal Heart Rate, Uterine Contractions, Fetal Distress, Oxygen Supply.

I. INTRODUCTION

The world's future relies on children, which is why it's crucial to ensure they are born without any complications or disabilities. Unfortunately, there are many new born infants who suffer from complex disorders like brain injury, cerebral palsy or even stillbirth caused by fetal distress, which occurs during delivery due to insufficient oxygen supply and other vital factors inside the mother's womb. To address this, medical professionals use cardiotocography, a widely used and effective method that analyses the fetal heart rate and the mother's uterine contractions. By analysing various fetal distress conditions and their possibilities, this method plays a significant role in identifying potential risks. Nowadays, though there are widespread uses of CTG, it suffers from inter-and intra-observer variation which results in false positives.

II. LITERATURE REVIEW

Sl. No.	Paper Title	Method	Limitations
1	Fetal Distress Classification using Cardiotocography	Random forests	Less amount of data and high number of features have been used
2	Fetal Health Classification from Cardiotocograph for Both Stages of Labor— A Soft-Computing- Based Approach	Random forests SVM MLP Bagging	Inconsistent Parameters
3	Multimodal Convolutional Neural Networks to Detect Fetal Compromise During Labor and Delivery	CNN	Labor is divided into 2 stages, the 2nd stage does not take 1st stage data into consideration.
4	Cardiotocography Analysis Using Conjunction of Machine Learning Algorithms	Decision Trees Support Vector Machines (SVM) Random Forests Neural Networks Gradient Boosting	Inconsistent Parameters

5	Fetal Health Classification Based on Machine Learning	Gradient Boosting Cat Boost Light Gradient Boosting Machine Cascade Forest Classifier	Lacking Feature Extraction algorithm
6	Decision Tree to Analyze the Cardiotocogram Data for Fetal Distress Determination	Decision Trees	Class Imbalance Problem
7	Fetal Heart Baseline Extraction and Classification based on Deep Learning	LTSM Models	Missing an automating scoring system to quantify mortality risk of the baby for better complication awareness and small amount of data is used
8	Prediction of Fetal Distress Using Linear and Non-linear Features of CTG Signals	Feature Selection Algorithms with the involvement of Decision Trees and KNN	Referring the costliness of using deep learning problems increasing the usage of hardware resources to run algorithms.

Despite the worldwide acceptance of CTG, the medico-legal issues have risen due to reasons such as improper interpretation of CTG and subsequent lack of timely action.

The lack of standard guidelines for interpretation and recognition of the FHR signals in the grey zone is another reason for misinterpretation. To address these issues, in recent times ML-based methods have been explored to find better interpretation of FHR results, which yielded results that are comparable to the clinical interpretation. These systems are capable of distinguishing between normal and abnormal fetuses.

The authors have decided to use an actual CTG Realtime data consisting of Fetal Heart Rate and Uterine contraction and extract minimal features from the recorded real time data and performing classification on several algorithms in the first stage of labor based on the two standard parameters which are: Fetal Heart Rate and Uterine Contractions and classify if it's a positive case or a negative case and choose the best optimal method.

The method would help a better interpretation of the CTG data and can provide more confidence in providing an accurate diagnosis of a case.

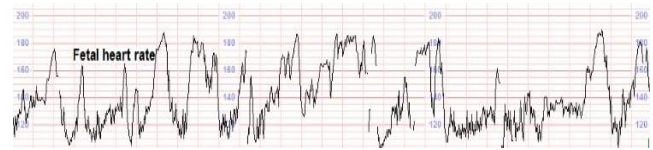


Figure 3.i: A CTG graphical data consisting of Fetal Heart Rate Data.

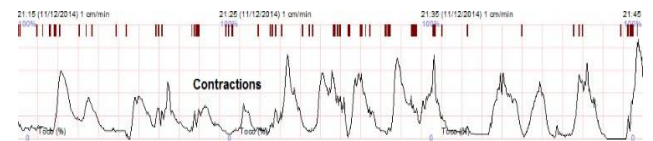


Figure 3.ii: A real time CTG graphical data consisting of Uterine Contractions data.

- Extracting the accurate real-time CTG graphical data and converting into numerical values and use it in models and choosing a model with most optimum result which can provide the most clarity on the cases.
- Try to solve the class imbalance and provide better generalization to the model.
- Finding patterns in data to find and maximize these results to find the best possible diagnosis of the case.
- Using the waveform database (wfdb) package to extract signals and convert them to numerical forms.
- Building a very simple algorithm to implement the cardiotocography (ctg) dataset.

IV. METHODOLOGY

a. Data Collection:

1. The `wfdb` module is imported, which is used for reading and working with PhysioNet's WFDB files.

2. The `Path` module from `pathlib` is imported to handle file paths.
3. The `os` module is imported to interact with the operating system.
4. The `numpy` module is imported to work with arrays and numerical operations.
5. The `pyplot` module from `matplotlib` is imported to create plots.
6. A message is printed indicating the total number of CTG recordings.
7. The `pandas` module is imported to work with data in tabular form.
8. The labels are read from a CSV file using the `read_csv` function and stored in the `labels` variable.
9. The `rec_id` column from the `labels` DataFrame is converted into a list and printed.
10. An informational message is printed regarding the annotations in the annotation file.
11. The file paths of the CTG recordings are appended to the `path_of_only_fnames` list.
12. The `ctg_collection` variable is assigned the value of `path_of_only_fnames`.
13. The variable `test_record` is assigned the value of the first record in `ctg_collection`.
14. Information about the sample record is printed.
15. The `rdsamp` function from `wfdb.io` is used to read the `test_record` and store the result in `test_record` variable.
16. Information about the record being displayed is printed.
17. A sample record from the CTG data is displayed.

b. Data Pre-Processing:

0. `ctg_dataframe` is a DataFrame containing the CTG (Cardiotocography) data for a specific patient. Let's go through the code step by step:
1. `x` is assigned the value of `ctg_dataframe.iloc[0,0]`. This code selects the first row (0) and the first column (0) of the `ctg_dataframe` DataFrame, extracting a specific portion of the data corresponding to patient 1001.
2. The `rename` function is called on `x` to rename the column labels. The code `columns={0: "FHR", 1: "UC"}` assigns the names "FHR" and "UC" to the first and second columns of `x`, respectively. This step provides meaningful names to the columns for better interpretation.
3. A message is printed indicating that the following DataFrame representation is about Patient 1001.
4. Finally, `x` is printed, displaying the updated DataFrame representation of the CTG data for Patient 1001. The DataFrame now has column labels "FHR" (representing Fetal Heart Rate) and "UC" (representing Uterine Contractions).

c. Data Visualization:

In this code snippet, the Matplotlib library is used to create two subplots and plot the CTG data for Patient 1001

The first line sets the figure size to (20, 18) inches using the `figure()` function from Matplotlib. This ensures that the resulting plot is large and easily readable.

The `subplot()` function is called with parameters (2, 1, 1). This creates a grid of subplots with 2 rows, 1 column, and selects the first subplot.

The `plot()` function is used to plot the "FHR" (Fetal Heart Rate) column from the DataFrame `x`. The option `--` specifies that a dashed line should be used for the plot, and the color is set to red ("r").

The `ylabel()` function is called to set the label for the y-axis as "Fetal Heart Rate".

The `title()` function is used to set the title of the entire plot as "CTG plot".

Another `figure()` function is called to create a new figure with the same size as the previous one.

The `subplot()` function is called again, this time with parameters (2, 1, 2). This selects the second subplot in the grid.

The `plot()` function is used to plot the "UC" (Uterine Contractions) column from the DataFrame `x`. The `ls` parameter is set to `--`, specifying a dashed line style.

The `ylabel()` function sets the label for the y-axis as "Uterine Contraction Spikes".

The `xlabel()` function sets the label for the x-axis as "Time Stamps".

Finally, the `show()` function is called to display the plot with both subplots.

Overall, this code generates a plot with two subplots, one showing the Fetal Heart Rate over time and the other showing the Uterine Contraction Spikes over time for Patient 1001.

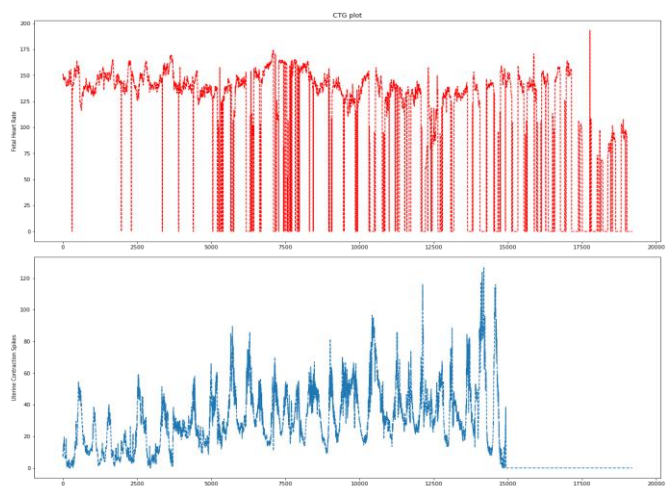


Fig IV.c.i: Plots of Fetal Heart Rate (FHR) and Uterine Contractions (UC).

d. Statistical Analysis of Data:

The code `x.describe()` calculates and displays descriptive statistics of the DataFrame `x`. The `describe()` function provides a summary of the central tendency, dispersion, and shape of the distribution of each column in the DataFrame. Let's go through the output that would typically be displayed:

1. Count: The count represents the number of non-null values in each column.
2. Mean: The mean is the average value of each column.
3. Standard Deviation: The standard deviation measures the dispersion or variability around the mean. It indicates how much the values in each column deviate from the mean.
4. Minimum: The minimum value in each column.
5. 25th Percentile (First Quartile): This value represents the point below which 25% of the data falls.
6. 50th Percentile (Median or Second Quartile): The median is the value separating the higher half from the lower half of the data.

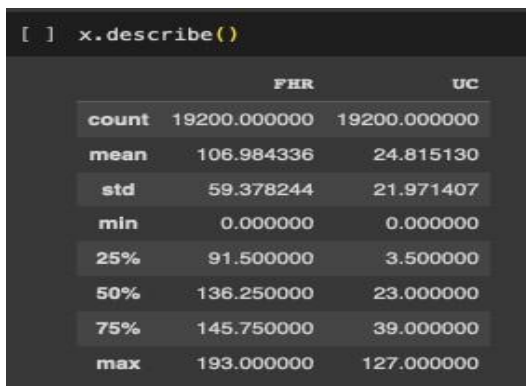
7. 75th Percentile (Third Quartile): This value represents the point below which 75% of the data falls.
8. Maximum: The maximum value in each column.

The ``describe()`` function provides a quick overview of the statistical properties of the numerical columns in the DataFrame ``x``, such as the range, distribution, and spread of the data. By calling ``x.describe()``, you can obtain these statistics for the "FHR" (Fetal Heart Rate) and "UC" (Uterine Contraction) columns.

The provided code snippet involves the processing and extraction of additional information from the CTG records. Let's break down the code step by step:

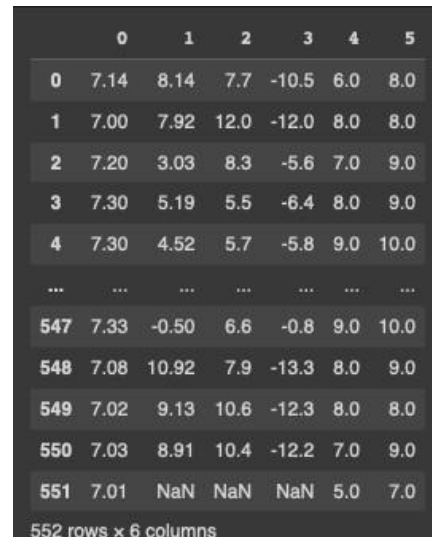
1. ``lst_parameter`` is a list comprehension that iterates over ``ctg_collection``, and for each CTG record, it calls ``wfdb.io.rdsamp(i)[1]`` to extract the parameter information. This information is then stored in the ``lst_parameter`` list.
2. ``lst_parameter_dataframe`` is created as a DataFrame using ``pd.DataFrame(lst_parameter)``, where each row represents the parameter information for a specific CTG record.
3. ``ctg_dataframe["sig_len"]`` is assigned the values from the "sig_len" column of ``lst_parameter_dataframe``. This adds a new column named "sig_len" to ``ctg_dataframe`` containing the signal length information.
4. ``lst_parameter_dataframe["comments"]`` retrieves the comments from the first CTG record in the ``lst_parameter_dataframe``.
5. The variable ``outcomes`` is initialized as an empty list.
6. A loop iterates over the "comments" column of ``lst_parameter_dataframe``, extracting specific elements from each comment and appending them as a list to the ``outcomes`` list.
7. ``outcome_post_attributes`` is initialized as an empty list.
8. Another loop iterates over ``outcomes``, splitting each element and extracting the numerical values from the last part of each split, and appending them to the ``outcome_post_attributes`` list.
9. ``outcome_post_attributes`` is converted into a DataFrame named ``outcome_df``.

At this point, the ``outcome_df`` DataFrame should contain the extracted numerical values from the comments of each CTG record, representing certain outcome attributes.



	FHR	UC
count	19200.000000	19200.000000
mean	106.984336	24.815130
std	59.378244	21.971407
min	0.000000	0.000000
25%	91.500000	3.500000
50%	136.250000	23.000000
75%	145.750000	39.000000
max	193.000000	127.000000

Fig.IV.d.i: Statistical Analysis for the Patient '1001'.



	0	1	2	3	4	5
0	7.14	8.14	7.7	-10.5	6.0	8.0
1	7.00	7.92	12.0	-12.0	8.0	8.0
2	7.20	3.03	8.3	-5.6	7.0	9.0
3	7.30	5.19	5.5	-6.4	8.0	9.0
4	7.30	4.52	5.7	-5.8	9.0	10.0
...
547	7.33	-0.50	6.6	-0.8	9.0	10.0
548	7.08	10.92	7.9	-13.3	8.0	9.0
549	7.02	9.13	10.6	-12.3	8.0	8.0
550	7.03	8.91	10.4	-12.2	7.0	9.0
551	7.01	NaN	NaN	NaN	5.0	7.0

552 rows x 6 columns

Fig IV.d.ii. Outcome Attributes for different patients.

e. Combining of Dataframes:

The DataFrame ``final_extracted_data`` is created by concatenating three DataFrames: ``ctg_copy1``, ``outcome_df``, and ``label_df``. Let's break down the code:

1. ``final_extracted_data`` is assigned the result of concatenating the three DataFrames using the ``concat()`` function. The DataFrames are concatenated along the columns (axis=1) by specifying ``axis=1`` as the parameter.
2. The ``rename()`` function is called on ``final_extracted_data`` to rename the columns. The code ``columns={0: 'PH', 1: 'BDecf', 2: 'pCO2', 3: 'BE', 4: 'Apgar1', 5: 'Apgar5', 6: 'rec_id', 7: 'eval_step4'}`` assigns the specified column names to the respective columns in ``final_extracted_data``.
3. The ``inplace=True`` parameter is used to modify ``final_extracted_data`` in place, updating the column names. ``final_extracted_data`` should contain the concatenated data from ``ctg_copy1``, ``outcome_df``, and ``label_df``, with the columns renamed as specified.

In the provided code snippet, several lists (``mean_FHR1``, ``mean_UC1``, ``mean_FHR2``, ``mean_UC2``, ``mean_FHR3``, ``mean_UC3``, ``mean_FHR4``, ``mean_UC4``) are initialized. These lists will be populated with mean values calculated from the "Patient Attributes" column of the ``final_extracted_data`` DataFrame. Let's break down the code:

1. The ``for`` loop iterates over the elements of the "Patient Attributes" column in ``final_extracted_data``.
2. For each iteration, the element ``i`` is split into four equal-sized parts using ``np.array_split(i, 4)``. This splits the element ``i`` into four subarrays.
3. For each subarray, the ``np.mean()`` function is used to calculate the mean along the first axis (``axis=0``). This results in the mean values for the "FHR" (Fetal Heart Rate) and "UC" (Uterine Contraction) in each subarray.
4. The mean values for each subarray are appended to the respective lists (``mean_FHR1``, ``mean_UC1``, ``mean_FHR2``, ``mean_UC2``, ``mean_FHR3``, ``mean_UC3``, ``mean_FHR4``, ``mean_UC4``).

After executing this code, the lists `mean_FHR1`, `mean_UC1`, `mean_FHR2`, `mean_UC2`, `mean_FHR3`, `mean_UC3`, `mean_FHR4`, and `mean_UC4` should contain the mean values of the "FHR" and "UC" for each quarter of the "Patient Attributes" column in `final_extracted_data`. These lists are calculated separately for each quarter of the column.

a DataFrame named `stage_data` is created using the lists `mean_FHR1`, `mean_UC1`, `mean_FHR2`, `mean_UC2`, `mean_FHR3`, `mean_UC3`, `mean_FHR4`, and `mean_UC4`. The `rename()` function is then called to rename the columns of `stage_data`. Let's break down the code:

1. The `pd.DataFrame()` function is used to create a DataFrame named `stage_data`. The data for the DataFrame is provided as a list containing the lists `mean_FHR1`, `mean_UC1`, `mean_FHR2`, `mean_UC2`, `mean_FHR3`, `mean_UC3`, `mean_FHR4`, and `mean_UC4`. The `transpose()` function is called to transpose the DataFrame so that the lists become columns.
2. The `rename()` function is called on `stage_data` to rename the columns. The `columns_stage` variable (not provided in the code snippet) should contain a dictionary mapping the current column names to the desired new column names. The code `columns_stage` is replaced with the actual dictionary you have in your code.
3. The `inplace=True` parameter is used to modify `stage_data` in place, updating the column names.

After executing this code, `stage_data` should be a DataFrame containing the mean values of "FHR" and "UC" for each quarter of the "Patient Attributes" column in `final_extracted_data`. The columns of `stage_data` will be renamed according to the mappings specified in the `columns_stage` dictionary. and `stage_data` is a DataFrame containing the mean values of "FHR" and "UC" for different stages.

	mean_FHR1	mean_UC1	mean_FHR2	mean_UC2	mean_FHR3	mean_UC3	mean_FHR4	mean_UC4
0	142.791771	18.046146	121.033906	35.851250	97.874740	41.920625	66.236927	3.442500
1	149.702917	30.352396	148.808073	30.458125	122.942917	33.400625	66.802396	4.899271
2	78.515833	1.177778	105.027944	20.011111	122.382000	30.128889	82.592278	8.253333
3	150.223393	34.329524	139.358214	15.879048	128.696429	13.567143	113.842262	24.563333
4	57.327222	7.748111	123.351667	16.589333	103.951889	24.530000	85.558722	12.475778
...
547	123.485800	52.159505	119.731188	59.809369	117.615671	47.081237	2.320807	0.005031
548	97.520392	22.895588	98.534167	9.580000	97.381127	31.380882	37.733627	8.380588
549	153.962537	30.367936	149.598629	21.091984	154.524126	10.336194	2.237126	0.372828
550	84.898362	16.688233	96.913265	9.213749	124.058421	18.835348	78.635039	11.470588
551	71.783490	20.268443	105.588054	42.557274	94.063023	25.187709	0.991378	0.029473

Fig.IV.e.i: Mean Data has been divided to 4 stages.

f. Construction of Co-relation Matrix:

A correlation matrix is calculated for the DataFrame `model_df`. Let's break down the code:

1. The `corr()` function is called on `model_df` to calculate the correlation between columns. The `corr()` function computes the pairwise correlation of columns using the default method, which is Pearson correlation by default.
2. The resulting correlation matrix is assigned to the variable `df_corr`.

After executing this code, `df_corr` should be a DataFrame representing the correlation matrix of `model_df`. Each value in the matrix represents the correlation between two columns in `model_df`, ranging from -1 to 1. A value close to 1 indicates a strong positive correlation, a value close to -1 indicates a strong negative correlation, and a value close to 0 indicates no or weak correlation.

You can further analyze the correlation matrix to gain insights into the relationships between variables in `model_df`.

g. Model Building using KNN:

The `StandardScaler` class from the `sklearn.preprocessing` module is imported. Additionally, the `train_test_split` function from the `sklearn.model_selection` module is imported. The code then performs feature scaling and prepares the input features (X) and target variable (y) for a machine learning model. Let's break down the code:

1. The `StandardScaler` class is imported from `sklearn.preprocessing`. This class is used for feature scaling, which transforms the data to have zero mean and unit variance.
2. The `train_test_split` function is imported from `sklearn.model_selection`. This function is commonly used to split the dataset into training and testing subsets.
3. The `scaling` object is created as an instance of the `StandardScaler` class.
4. The `fit_transform()` method of the `scaling` object is called on a subset of `model_df`. The `iloc[:,0:8]` indexing selects the first 8 columns of `model_df` as the input features. The data in these columns is then transformed using the `StandardScaler`, and the result is assigned to the variable `X`.
5. The `y` variable is assigned the values from the "eval_step4" column of `model_df`. This column represents the target variable.

After executing this code, the input features `X` will contain the scaled values of the selected columns from `model_df`, and the target variable `y` will contain the values from the "eval_step4" column. You can use `X` and `y` to train and evaluate in KNN.

	mean_FHR1	mean_UC1	mean_FHR2	mean_UC2	mean_FHR3	mean_UC3	mean_FHR4
mean_FHR1	1.000000	0.850023	0.848097	0.838840	0.855851	0.835846	0.846235
mean_UC1	0.850023	1.000000	0.861945	0.856041	0.832020	0.835226	0.855385
mean_FHR2	0.848097	0.861945	1.000000	0.847703	0.807972	0.848544	0.863365
mean_UC2	0.838840	0.856041	0.847703	1.000000	0.827910	0.870677	0.845183
mean_FHR3	0.855851	0.832020	0.807972	0.827910	1.000000	0.830293	0.811759
mean_UC3	0.835846	0.835226	0.848544	0.870677	0.830293	1.000000	0.850106
mean_FHR4	0.846235	0.855385	0.863365	0.845183	0.811759	0.850106	1.000000

Fig.IV.e.ii: Co-relation matrix of the stage data.

h. Training and Testing the Model:

The dataset is split into training and testing subsets using the `train_test_split` function from `sklearn.model_selection`. The K-nearest neighbors (KNN) classification model is then trained using the training data, and its accuracy is evaluated on the testing data. Additionally, cross-validation is performed using the `cross_val_score` function to assess the model's performance. Let's break down the code:

1. The `train_test_split` function is used to split the data into training and testing subsets. The input features `X` and target variable `y` are passed as parameters, along with the `test_size` parameter set to 0.20, indicating that 20% of the data should be used for testing. The `random_state` parameter is set to 42 to ensure reproducibility.
2. The resulting training and testing subsets are assigned to `X_train`, `X_test`, `y_train`, and `y_test`, respectively.
3. The `neighbors` module from `sklearn` is imported to use the K-nearest neighbors classification algorithm.
4. An instance of the `KNeighborsClassifier` class is created with `n_neighbors=4`, indicating that the model will consider the labels of the four nearest neighbors when making predictions.
5. The `fit()` method is called on the `beta_model` object, using `X_train` as the input features and `y_train` as the target variable, to train the KNN model.
6. The `score()` method is called on the `beta_model` object, using `X_test` and `y_test`, to evaluate the accuracy of the trained model on the testing data. The result is assigned to `beta_scores`.
7. The accuracy of the beta model is printed, displaying the value of `beta_scores` as a percentage.
8. The `cross_val_score()` function is used to perform cross-validation. The KNN model with `n_neighbors=4` is passed as the estimator, along with the training data (`X_train` and `y_train`), `cv=10` indicating 10-fold cross-validation, and `scoring='accuracy'` specifying that accuracy should be used as the evaluation metric.
9. The resulting cross-validation scores are assigned to `scores`.

After executing this code, you should have the following results:

- The accuracy of the beta model on the testing data is printed as a percentage.
- The `scores` variable will contain an array of accuracy scores obtained from the cross-validation.

These results provide an assessment of the model's performance on both the testing data and through cross-validation.

The cross-validation scores obtained from the `cross_val_score` function are printed along with the mean validation score. Let's break down the code:

1. The code initiates a loop that iterates over the range of the length of the `scores` array.
2. Within each iteration, the current validation score is printed using the `print()` function. The format specifier `{}` is used to indicate the placeholder for the validation index and score values.
3. The validation index is incremented by 1 in the output using `i+1`.
4. The mean validation score is calculated using the `np.mean()` function, passing `scores` as the input array.
5. The mean validation score is printed using the `print()` function, along with an appropriate message.

After executing this code, you should see the following output:

- The individual validation scores for each fold of the cross-validation are printed.
- The mean validation score is printed.

i. Demonstration of the Model:

Finally the trained K-nearest neighbors (KNN) classification model `beta_model` is used to make predictions on a new input `x_input`. Based on the predicted value, different messages are printed to describe the condition of the fetus. Let's break down the code:

1. The `predict()` method is called on the `beta_model` object, passing `x_input.reshape(1,8)` as the input. The `reshape(1,8)` is used to reshape the `x_input` array into a 2D array with a single sample and 8 features.
2. The predicted value is extracted from the resulting array using `[0]` indexing and assigned to the variable `value`.
3. Conditional statements (`if` statements) are used to check the value of `value` and print corresponding messages based on the condition.
 - If `value` is equal to 1, it means the fetus is predicted to be suffering from no fetal distress, and the corresponding message is printed.
 - If `value` is equal to 2, it means the fetus is predicted to be suffering from mild fetal distress, and the corresponding message is printed.
 - If `value` is equal to 3, it means the fetus is predicted to be suffering from severe fetal distress and further investigation is needed. Two messages are printed in this case:
 - If `value` is equal to -1, it indicates that the record is uninterpretable"

After executing this code, based on the predicted value from the KNN model, the appropriate message will be printed to describe the condition of the fetus.

EXPERIMENTAL DETAILS

Languages used are Python:

- Python Libraries for data- pre-processing, machine learning algorithm, visualization and statistical analysis.
- Python IDE used is Google Colab mainly for cloud and hardware resources.
- Realtime Stage-1 Labor CTG Data.

OUTCOMES

- To make a good, generalized model, so there's an optimal bias-variance trade-off and solve the class-imbalance problem.
- Feed new real time data and simultaneously use feature extraction algorithms to make the process automated data.
- Providing doctor, a summary of medical analysis which are automated by the algorithm.

- Simpler data extraction has been achieved using the wfdb package rather than building a complicated data extraction tool.

RESULTS

The accuracy of the beta model is : 89.54954954954954%,

The following cross validation scores are:

Validation 1 is 0.80555556%

Validation 2 is 0.86363636%

Validation 3 is 0.81818182%

Validation 4 is 0.90909091%

Validation 5 is 0.7272727299999999%

Validation 6 is 0.88636364%

Validation 7 is 0.75%

Validation 8 is 0.81818182%

Validation 9 is 0.84090909%

Validation 10 is 0.79545455%

As we converted our signals into 4 sub stages in Stage-1 and Stage-2,. For our paper we are focusing on Stage-1 data to predict the probability of Fetal Distress before going to Stage 2

CONCLUSION

As we have less data, 552 records. In the future, we hope this work could be extended and the model be run on high volume of data. To avoid overfitting and get good generalization throughout the data.

ACKNOWLEDGMENT

I would like to thank Presidency University for their timely help via discussions, presentations and communications and especially to my Supervisor Prof. Dr. Sulaiman Syed Mohammed for his input.

REFERENCES

1. Artificial neural networks applied to fetal monitoring in labour 22:85–93 DOI 10.1007/s00521-011-0743-y (2013)
2. Cardiotocography Analysis Using Conjunction of Machine Learning Algorithms DOI 10.1109/CMVIT.2017.27978-1-5090-4993-6/17
3. Decision Tree to Analyze the Cardiotocogram Data for Fetal Distress Determination 978-1-5386-2182-0/17/\$31.00 ©2017 IEEE
4. Multimodal Convolutional Neural Networks to Detect Fetal Compromise During Labor and Delivery DOI:10.1109/ACCESS.2019.2933368
5. Fetal Heart Baseline Extraction and Classification based on Deep Learning 978-1-7281-6494-6/19/\$31.00 ©2019 IEEE DOI 10.1109/ITCA49981.2019.00053
6. Cardiotocographic Signal Feature Extraction Through CEEMDAN and Time-Varying Autoregressive Spectral-Based Analysis for Fetal Welfare Assessment
DOI 10.1109/ACCESS.2019.2950798
7. Prediction of Fetal Distress Using Linear and Non-linear Features of CTG Signals DOI: 10.1007/978-3-030-37218-7_5
8. FETAL DISTRESS CLASSIFICATION USING CARDIOTOCOGRAPHY www.jetir.org (ISSN-2349-5162)
9. Fetal Health Classification Based on Machine Learning 978-1-6654-1540-8/21/\$31.00 ©2021 IEEE
10. Fetal Health Classification from Cardiotocograph for Both Stages of Labor—A Soft-Computing-Based Approach <https://doi.org/10.3390/diagnostics13050858>
11. Cardiotocography (ctg) dataset: <http://people.ciirc.cvut.cz/~spilkjir/data.html>