Lakehead
UNIVERSITY

Machine Learning Models for Early Detection of Fetal Distress

Amirhossein Zeinali Dehaghani

Narjes Atashinsima

FALL 2024

Contents

Abstract

According to the World Health Organization, there are nearly 2 million stillbirths[1] The most common methods of diagnosing perinatal death and taking early precaution for maternal and fetal health is NST (nonstress test) which is a pregnancy screening that measures fetal heart rate and reaction to movement.[1] This project aims on implementing predictive models for the early detection of fetal distress.

1 Introduction

1.1 Background

The NST measures the heart rate of the fetus in response to its own movements. The NST device is typically connected to a pregnant woman for 20 min, with two separate probes recording data on fetal heart rate (FHR) and uterine contractions (UCs).

1.2 Motivation

Every year, Over 40% of all stillbirths occur during labour, a loss that could be avoided with improved quality and respectful care during childbirth including routine monitoring and timely access to emergency obstetric care when required[2]

1.3 Objective

This project focuses on implementing predictive models for the early detection of fetal distress and aims to classify fetal health categories.

1.4 Scope

The dataset used in this project is a labeled data of fetal health obtained from the CTG [2] data of 2126 pregnant women. There are 21 independent variables and 1 dependent variable which is tagged as 1 (Normal), 2 (Suspect) and 3 (Pathological). In this project, KNN, Decision Tree, and Random Forest are implemented after identifying and extracting features using exploratory data analysis (EDA) and principal component analysis (PCA). The models are evaluated with confusion matrix to assess classification performance.

[1]A baby who dies after 28 weeks of pregnancy, but before or during birth, is classified as a stillbirth.

[2]Cardiotocography (CTG) is a continuous recording of the fetal heart rate

2 Problem Definition

2.1 Problem Statement

This project addresses the critical issue of fetal distress, the possible condition where the fetus is not receiving enough oxygen which can lead to severe complications during childbirth.

2.2 Hypothesis

This project aims to use machine Learning Models for Early Detection of Fetal Distress. In order to experiment different classifiers, we use RF, DT and KNN algorithms for predicting fetal health and after that we evaluate the model's performance providing the accuracy score and confusion matrix.

2.3 Research Questions

- Which features in the fetal health dataset are most important about detection the fetal distress?
- How do implemented classification algorithms compare in terms of accuracy, precision, and recall for predicting fetal health categories?

3 Data Description

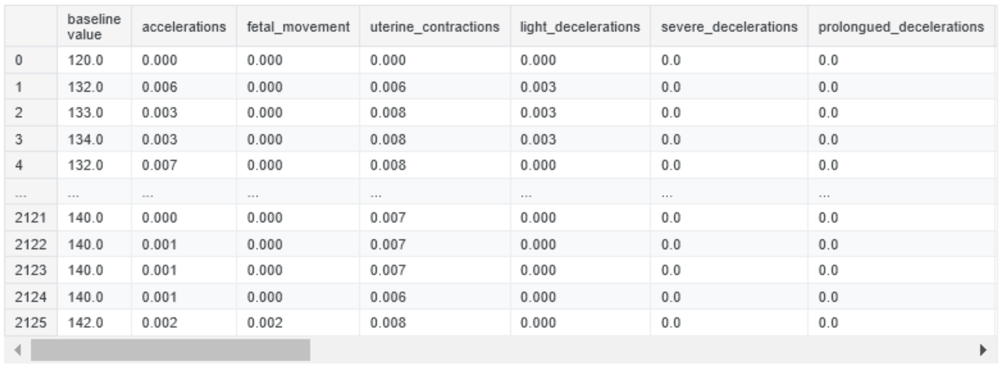
3.1 Dataset Source

The "Fetal Health Detection" dataset using in this project is obtained from Kaggle [4] and first step is to get familiar with the dataset and preprocess it to get more accuracy in algorithm results.

3.1.1 Loading Dataset

The dataset used in this project is a labeled data of fetal health obtained from the CTG data of 2126 pregnant women. There are 21 independent variables and 1 dependent variable.

```
fetal_data=pd.read_csv('../input/fetal-health-classification/fetal_health.csv')
fetal_data
```



	baseline value	accelerations	fetal_movement	uterine_contractions	light_decelerations	severe_decelerations	prolongued_decelerations
0	120.0	0.000	0.000	0.000	0.000	0.0	0.0
1	132.0	0.006	0.000	0.006	0.003	0.0	0.0
2	133.0	0.003	0.000	0.008	0.003	0.0	0.0
3	134.0	0.003	0.000	0.008	0.003	0.0	0.0
4	132.0	0.007	0.000	0.008	0.000	0.0	0.0
...	...	...	...	...	...	...	...
2121	140.0	0.000	0.000	0.007	0.000	0.0	0.0
2122	140.0	0.001	0.000	0.007	0.000	0.0	0.0
2123	140.0	0.001	0.000	0.007	0.000	0.0	0.0
2124	140.0	0.001	0.000	0.006	0.000	0.0	0.0
2125	142.0	0.002	0.002	0.008	0.000	0.0	0.0

Figure 1: Loading Dataset

3.2 Dataset Structure

3.2.1 Shape of dataset

Based on output, there are 2126 rows and 22 columns in the data set.

```
fetal_data.shape
```

output:

(2126, 22)

3.2.2 Features/columns Description

There are 21 independent variables (features) and 1 dependent variable (target value) as below:

- **Features: 'baseline value'** FHR baseline (beats per minute)

  'accelerations' Number of accelerations per second

  'fetal_movement' Number of fetal movements per second

  'uterine_contractions' Number of uterine contractions per second

  'light_decelerations' Number of light decelerations per second

  'severe_decelerations' Number of severe decelerations per second

  'prolongued_decelerations' Number of prolonged decelerations per second

  'abnormal_short_term_variability' Percentage of time with abnormal short term variability

  'mean_value_of_short_term_variability' Mean value of short term variability

  'percentage_of_time_with_abnormal_long_term_variability' Percentage of time with abnormal long term variability

  'mean_value_of_long_term_variability' Mean value of long term variability

  'histogram_width' Width of FHR histogram

  'histogram_min' Minimum (low frequency) of FHR histogram

'histogram_max' Maximum (high frequency) of FHR histogram

'histogram_number_of_peaks' Number of histogram peaks

'histogram_number_of_zeroes' Number of histogram zeros

'histogram_mode' Histogram mode

'histogram_mean' Histogram mean

'histogram_median' Histogram median

'histogram_variance' Histogram variance

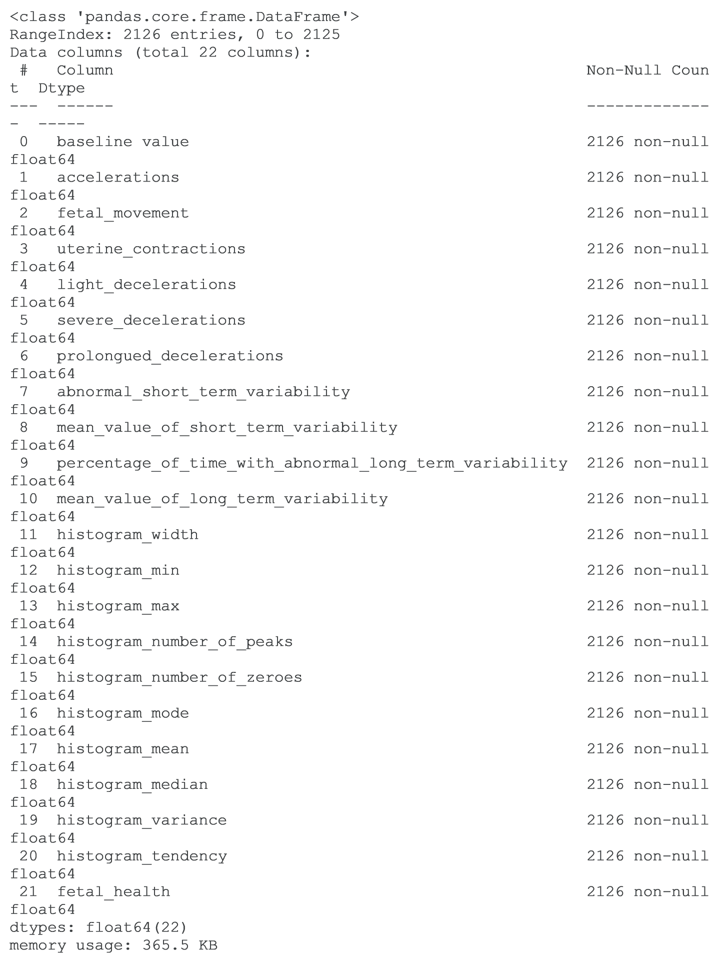
'histogram_tendency' Histogram tendency

- **Target:** fetal_health' Tagged as: 1.Normal 2. Suspect 3.Pathological

3.2.3 Type of values

Based on output, all the values in the dataset are float. There are no string values in the dataset.

```
fetal_data.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2126 entries, 0 to 2125
Data columns (total 22 columns):
 #   Column                                                  Non-Null Coun
t  Dtype
---  ------                                                  -------------
-  -----
 0   baseline value                                          2126 non-null
float64
 1   accelerations                                           2126 non-null
float64
 2   fetal_movement                                          2126 non-null
float64
 3   uterine_contractions                                    2126 non-null
float64
 4   light_decelerations                                     2126 non-null
float64
 5   severe_decelerations                                    2126 non-null
float64
 6   prolongued_decelerations                                2126 non-null
float64
 7   abnormal_short_term_variability                         2126 non-null
float64
 8   mean_value_of_short_term_variability                    2126 non-null
float64
 9   percentage_of_time_with_abnormal_long_term_variability  2126 non-null
float64
 10  mean_value_of_long_term_variability                     2126 non-null
float64
 11  histogram_width                                         2126 non-null
float64
 12  histogram_min                                           2126 non-null
float64
 13  histogram_max                                           2126 non-null
float64
 14  histogram_number_of_peaks                               2126 non-null
float64
 15  histogram_number_of_zeroes                              2126 non-null
float64
 16  histogram_mode                                          2126 non-null
float64
 17  histogram_mean                                          2126 non-null
float64
 18  histogram_median                                        2126 non-null
float64
 19  histogram_variance                                      2126 non-null
float64
 20  histogram_tendency                                      2126 non-null
float64
 21  fetal_health                                            2126 non-null
float64
dtypes: float64(22)
memory usage: 365.5 KB
```

Figure 2: Type of values

3.3 Data Cleaning/Preprocessing

3.3.1 Handling missing values

based on output, there are no null values hence the data is clean.

```
null=fetal_data.isnull().sum()
null
```

```
baseline value                                            0
accelerations                                             0
fetal_movement                                            0
uterine_contractions                                      0
light_decelerations                                       0
severe_decelerations                                      0
prolongued_decelerations                                  0
abnormal_short_term_variability                           0
mean_value_of_short_term_variability                      0
percentage_of_time_with_abnormal_long_term_variability    0
mean_value_of_long_term_variability                       0
histogram_width                                           0
histogram_min                                             0
histogram_max                                             0
histogram_number_of_peaks                                 0
histogram_number_of_zeroes                                0
histogram_mode                                            0
histogram_mean                                            0
histogram_median                                          0
histogram_variance                                        0
histogram_tendency                                        0
fetal_health                                              0
dtype: int64
```

Figure 3: Checking for missing values

3.3.2 Encoding categorical data

As we mentioned earlier and based on info(), There are no string or categorical values in this dataset and all the values are **float** and **continuous**.

3.3.3 Feature scaling

Mathematical behind the data Standardization Data Standardization plays an important role in order to make the features balanced and express the correlations in the model's train. In this project, we used standardization scaling which is a scaling method where the values are centered around the mean with a unit standard deviation.

$$x'_i = \frac{x_i - \mu}{\sigma} \qquad \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

Where, x_i is original feature value, μ is mean of the feature values, σ is standard deviation of the feature values, x'_i is scaled feature value, and n is the number of samples.

Feature scaling Implementation

Use boxplot in order to show the range of the feature attributes.

```
plt.figure(figsize=(20,10))
sns.boxenplot(data = fetal_data)
plt.xticks(rotation=90)
plt.show()
```

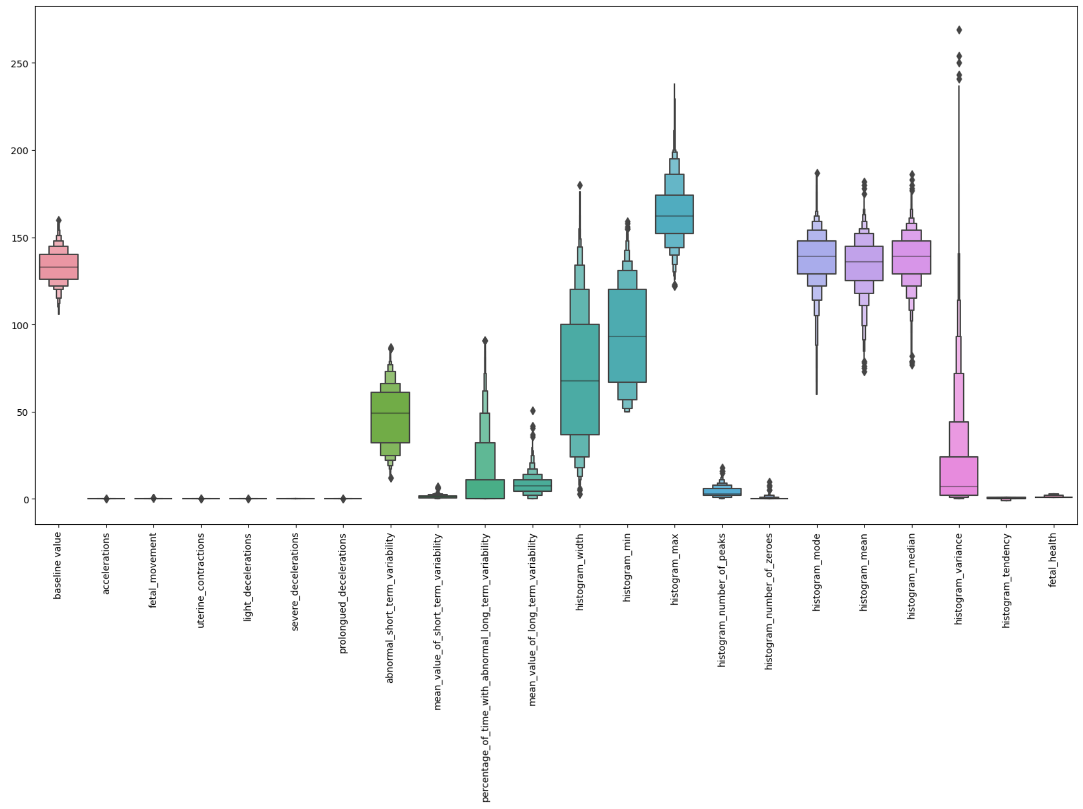


Figure 4: range of the features before scaling

Based on above boxplot, all the features are in different ranges. To fit this in a model we must scale it to the same range. Implement standardization in order to scale the data to have a mean of 0 and a standard deviation of 1:

```
scaler=StandardScaler()
scaler.fit(fetal_data)
scaled_data=scaler.transform(fetal_data)
scaled_data
```

Take a look on scaled data

```
plt.figure(figsize=(20,10))
sns.boxenplot(data = scaled_data)
plt.xticks(rotation=90)
plt.show()
```

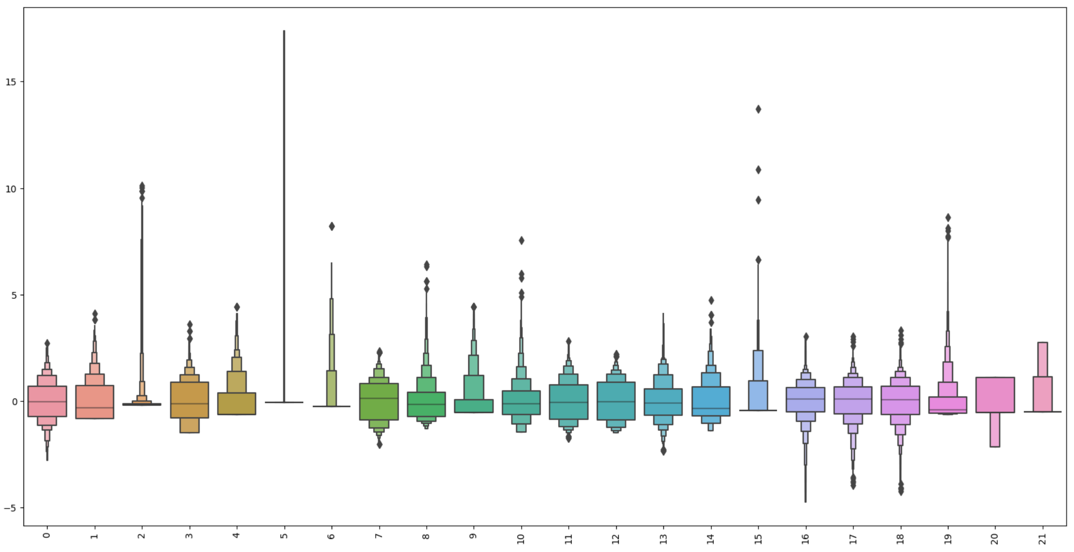


Figure 5: range of the features after scaling

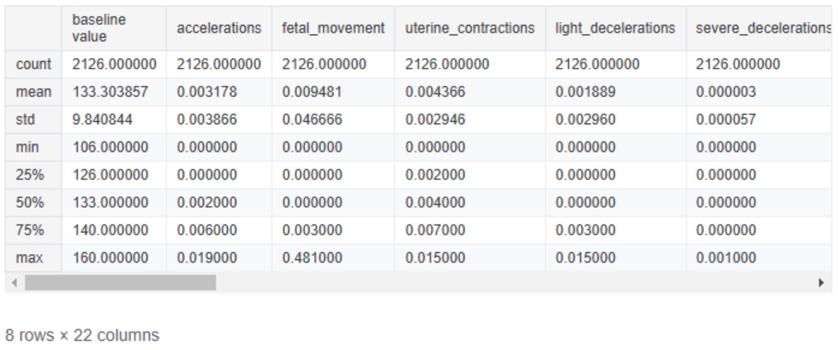
3.4 Exploratory Data Analysis (EDA)

In this part of the project,we first explored for the key statistics for the numerical features, including mean, median, standard deviation, minimum, and maximum are computed in order to review the data distribution and identify the spread of the features. In addition, various visualization methods such as correlation heat map, implot, box plot, etc. are performed to recognize the distributions and relationships between features and the target variable. Furthermore, bar plot is used to visualize the target column (fetal_health) to detect patterns of the fetal state (Normal, Suspect and Pathological).

3.4.1 Statistical summaries

describe() provides key metrics such as count, mean, standard deviation, minimum, and maximum values for every numerical column:

```
fetal_data.describe()
```



	baseline value	accelerations	fetal_movement	uterine_contractions	light_decelerations	severe_decelerations
count	2126.000000	2126.000000	2126.000000	2126.000000	2126.000000	2126.000000
mean	133.303857	0.003178	0.009481	0.004366	0.001889	0.000003
std	9.840844	0.003866	0.046666	0.002946	0.002960	0.000057
min	106.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	126.000000	0.000000	0.000000	0.002000	0.000000	0.000000
50%	133.000000	0.002000	0.000000	0.004000	0.000000	0.000000
75%	140.000000	0.006000	0.003000	0.007000	0.003000	0.000000
max	160.000000	0.019000	0.481000	0.015000	0.015000	0.001000

8 rows × 22 columns

Figure 6: out of describe()

3.4.2 Visualization

Target value analyzing

Use unique(), in order to find the uniqe values.

```
fetal_data['fetal_health'].unique()
```

output: array([2., 1., 3.])

The fetal health column as our target value, represented by 1, 2, 3 where, 1: normal, 2: suspect. 3: Pathological

Data visualizations of Target Value "fetal health"

```
sns.countplot(data= fetal_data, x="fetal_health")
```

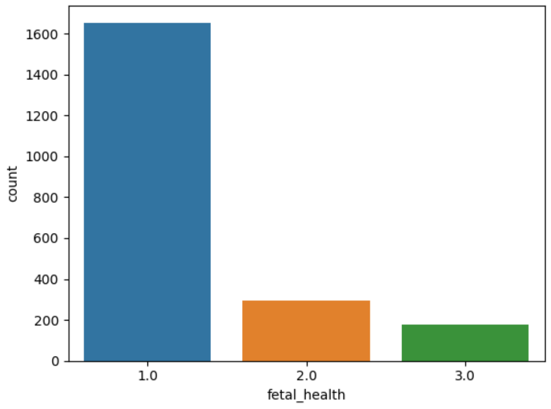


Figure 7: Target value analyzing

The figures shows that the number of 'Normal' cases is more followed by 'Suspect' cases followed by 'Pathological' cases.

Accelerations Vs Fetal Movement by Fetal Health

```
#Accelerations Vs Fetal Movement by Fetal Health
sns.lmplot(data =fetal_data,x="accelerations",y="fetal_movement",hue="fetal_health",
    legend_out=False)
plt.show()
```

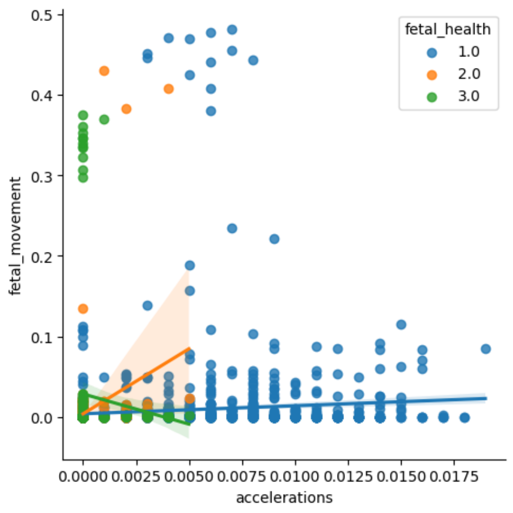


Figure 8: Accelerations Vs Fetal Movement by Fetal Health

Prolongued Decelerations Vs Fetal Movement by Fetal Health

```
#Prolongued Decelerations Vs Fetal Movement by Fetal Health
sns.lmplot(data =fetal_data,x="prolongued_decelerations",y="fetal_movement",hue="
    fetal_health",legend_out=False)
plt.show()
```

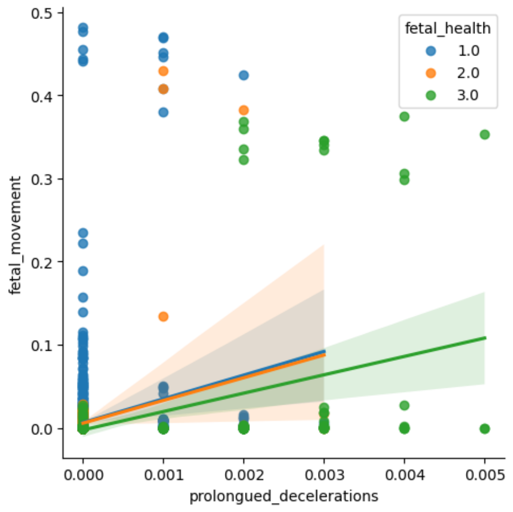


Figure 9: Prolongued Decelerations Vs Fetal Movement by Fetal Health

Abnormal Short Term Variability Vs Fetal Movement by Fetal Health

```
# Abnormal Short Term Variability Vs Fetal Movement by Fetal Health
sns.lmplot(data =fetal_data,x="abnormal_short_term_variability",y="fetal_movement",hue
    ="fetal_health",legend_out=False)
plt.show()
```

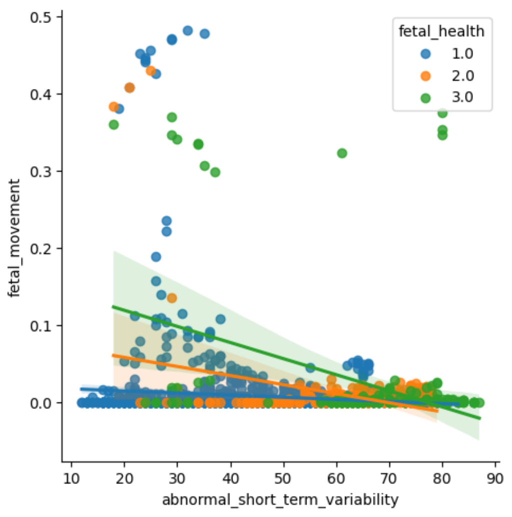


Figure 10: Abnormal Short Term Variability Vs Fetal Movement by Fetal Health

Mean Value Of Long Term Variability Vs Fetal Movement by Fetal Health

```
#Mean Value Of Long Term Variability Vs Fetal Movement by Fetal Health
sns.lmplot(data =fetal_data,x="mean_value_of_long_term_variability",y="fetal_movement"
    , hue="fetal_health",legend_out=False)
plt.show()
```

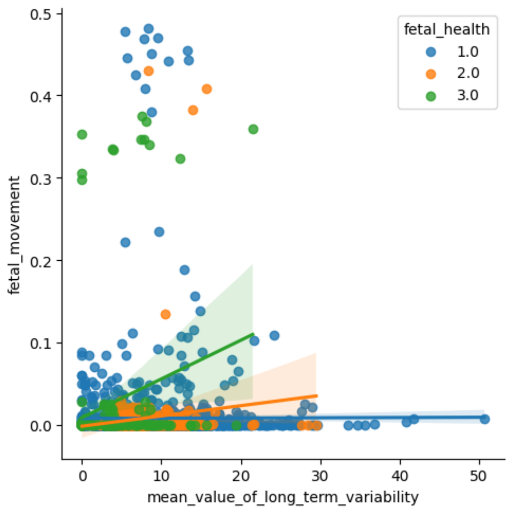


Figure 11: Mean Value Of Long Term Variability Vs Fetal Movement by Fetal Health

Output interpretation: Noticeably, The rates of change of the above-mentioned values with each target show a specific trend.

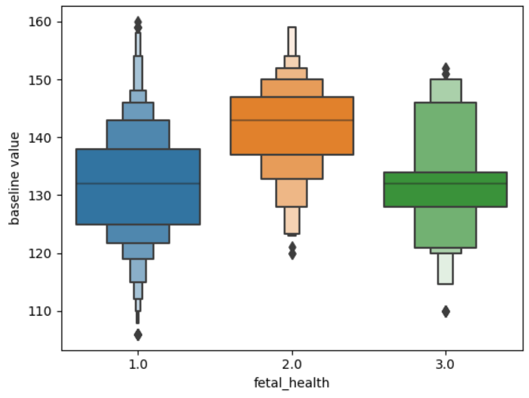
Outliers Identification The outliers on the dataset are spotted. However, it is not quite a reasonable idea to remove them as it may lead to over-fitting. Though we may end up with better statistics.

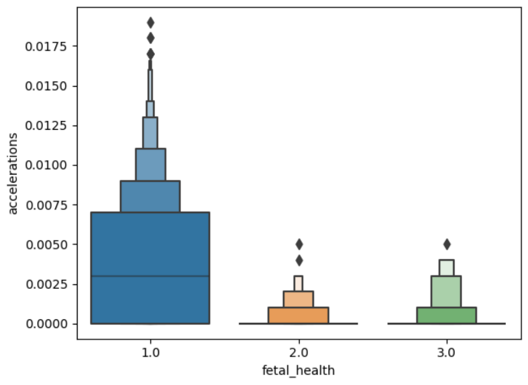
The basic rule of thumb for the outliers in question is: It is a measurement error or data entry error, correct the error if possible. If you can't fix it, remove that observation. In this case, this is the outcome of a CTG report so it is unlikely that this was a data entry error.

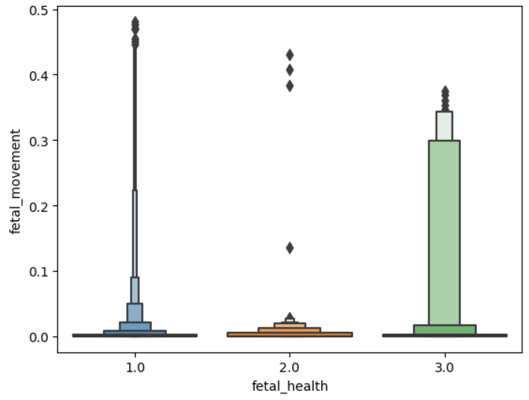
Thus, assuming that these are the natural part of the population we are studying, we should not remove it.

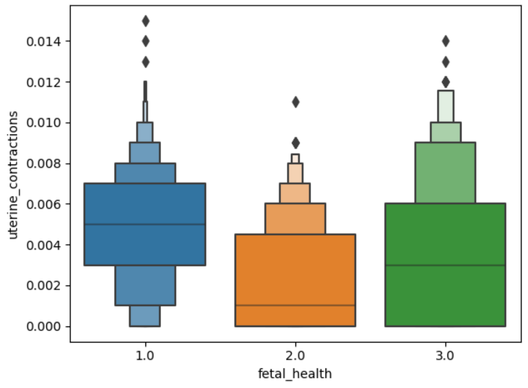
Feature analyzing using Boxplot

```
        'uterine_contractions', 'light_decelerations', 'severe_decelerations',
        'prolongued_decelerations', 'abnormal_short_term_variability',
        'mean_value_of_short_term_variability',
        'percentage_of_time_with_abnormal_long_term_variability',
        'mean_value_of_long_term_variability']
for i in cols:
    sns.boxenplot(x=fetal_data["fetal_health"], y=fetal_data[i])
    plt.show()
```









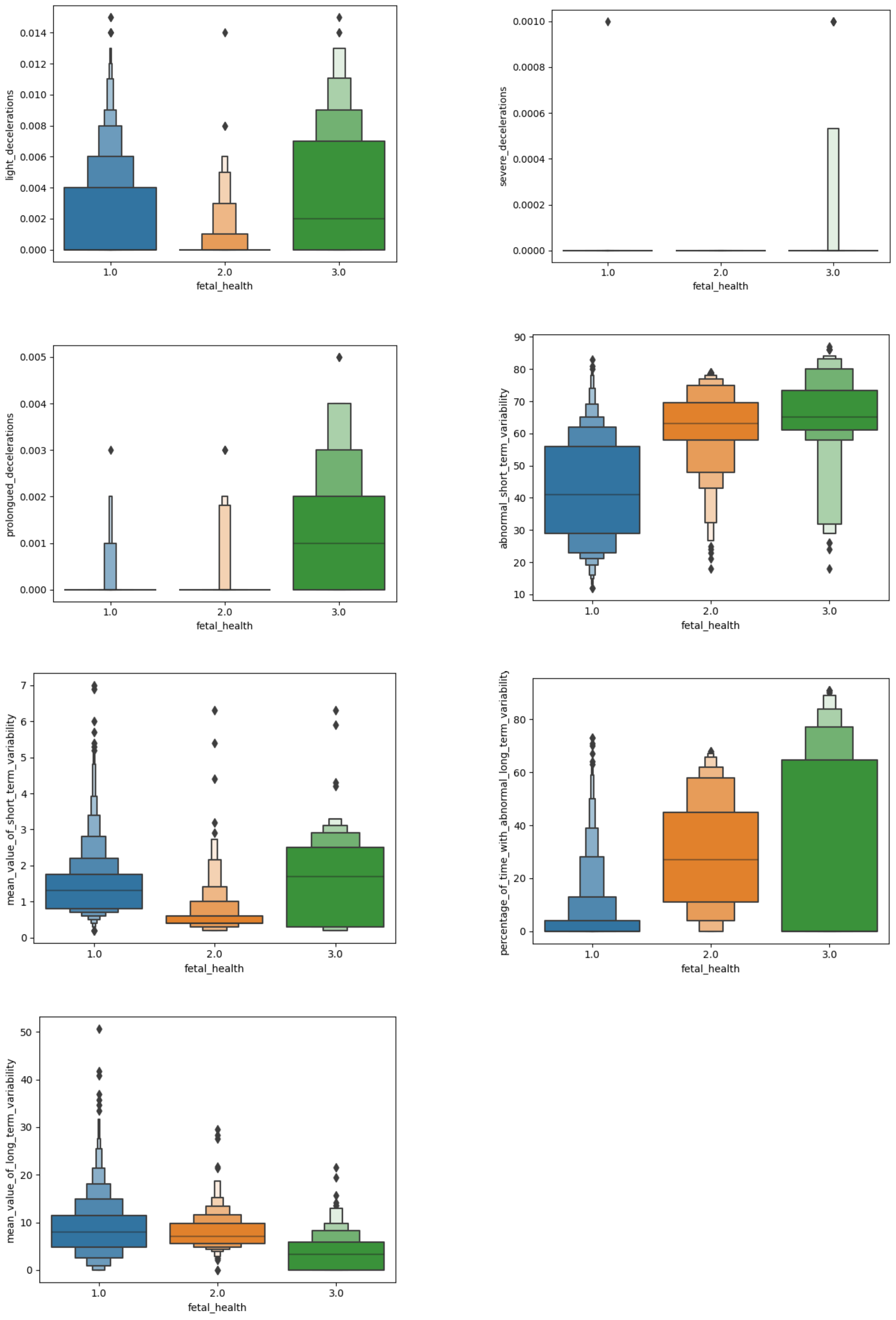


Figure 16: Feature analyzing using Boxplot

Feature analyzing using correlation

Correlation between features and the target "fetal health".

```python
numeric_data = fetal_data.select_dtypes(exclude="object")
numeric_corr = numeric_data.corr()
f,ax=plt.subplots(figsize=(25,1))
sns.heatmap(numeric_corr.sort_values(by=["fetal_health"], ascending=False).head(1),
    cmap="twilight")
plt.title("Numerical features correlation with the fetal_health", weight="bold",
    fontsize=18, color="#42A84F")
plt.yticks(weight="bold", color="orange", rotation=0)
plt.show()
```

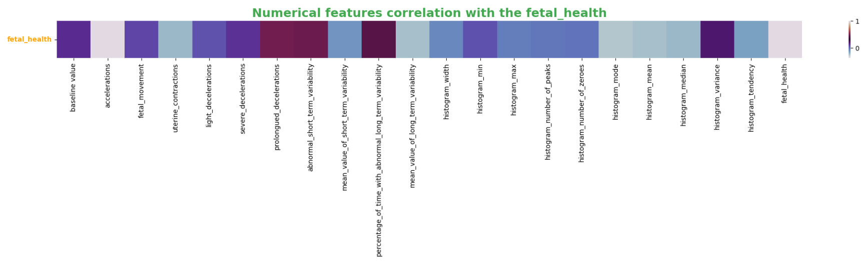


Figure 17: Feature analyzing using correlation

Output interpretation: we find the features "percentage_of_time_with_abnormal_long_term_variability", "abnormal_short_term_variability", and "prolongued_decelerations" as key factors that may influence fetal health positively, while "accelerations" negatively.

Feature analyzing using Correlation heatmap

```python
corr = fetal_data.corr()
plt.figure(figsize=(12,10))
sns.heatmap(corr, annot=True, cmap='gist_earth')
plt.show()
```

Output interpretation: The heatmap shows "histogram_mean" and "histogram_median" have the strongest correlation which is 0.95 and it means whenever one of them increases the other tends to increase too; and the second strongest correlation is between "histogram_median" and "histogram_mode" with the coefficient of 0.93. The most negative correlation with the coefficient of -0.9 is between "histogram_min" and "histogram_width", it means if one of them increases the other one decreases.

3.4.3 Dimensionality Reduction (Principle Components analysis)

The dataset includes various physiological measurements, such as fetal heart rate, accelerations, and uterine contractions, which can create high dimensionality. We used principle components analysis

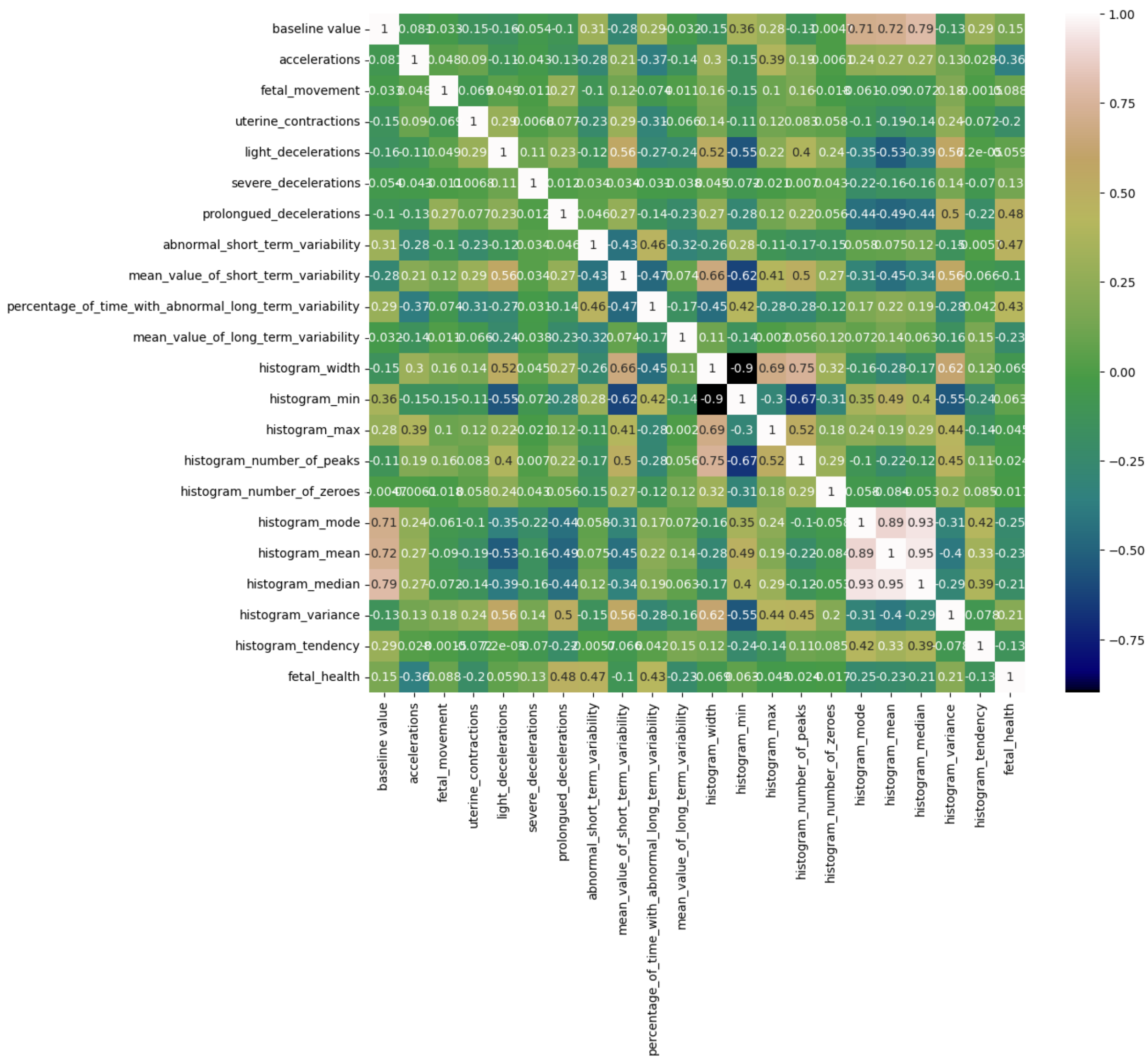


Figure 18: Feature analyzing using Correlation heatmap

(PCA) in order to reduces this dimensionality by transforming the standardized original features into a smaller set of variables (principal components) while essential variance within the data is preserved.

Principle Components analysis Implementation]

The dataset has been reduced into 14 components using principal component analysis.

```
pca=PCA(n_components=14)
pca.fit(scaled_data)
x_pca=pca.transform(scaled_data)
```

Take a look at the features after PCA

```
plt.figure(figsize=(20,10))
sns.boxenplot(data = x_pca)
```

```
plt.xticks(rotation=90)
plt.show()
```

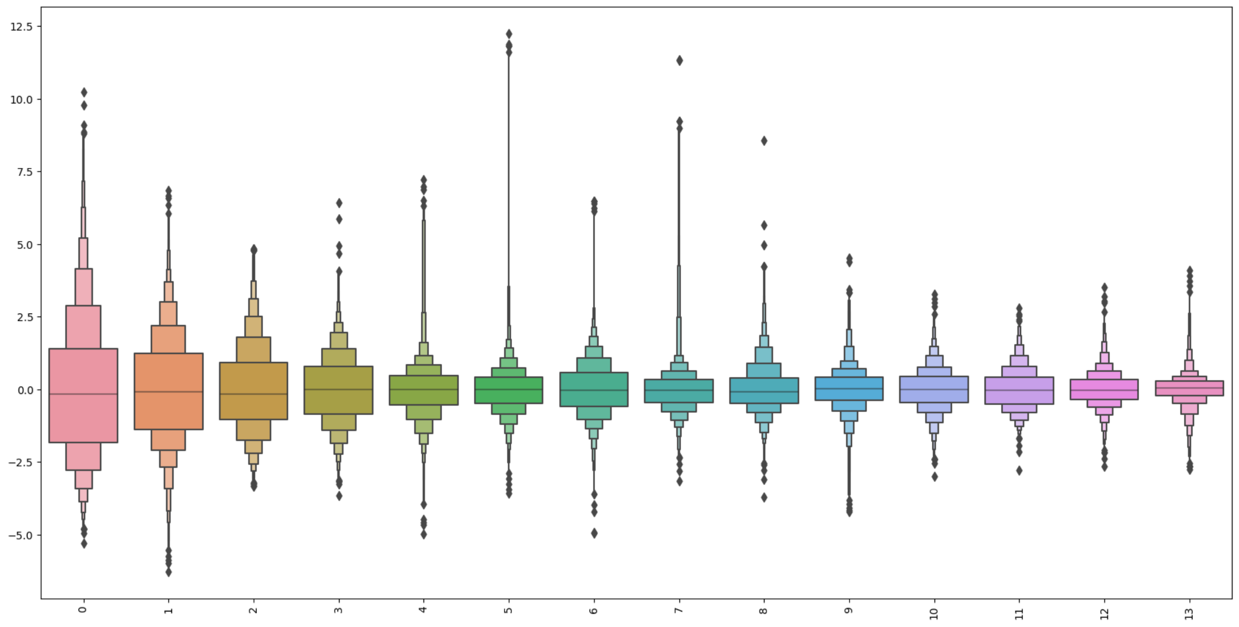


Figure 19: After PCA Implementation

4 Methodology

4.1 Algorithms/Models

In this part of the project, we describe the mathematical formulations for implemented classification algorithms and also model evaluation. Before detailing the algorithms. some general notation are defined in order to better understanding and preventing duplication of definitions.

- Input Features X: The input feature vector is consist of multiple physiological measurements.

$$X = [x_1, x_2, \dots, x_n]$$

- Class Label Y: The target variable is fetal health in this dataset and represent a multi-class variable:
  - 0: Normal
  - 1: Suspicious
  - 2: Pathological

- Classification Function $f(X)$: The function that predict class label Y over feature vector X.

$$f : X \to Y$$

4.1.1 Mathematical Formulation for Decision Tree

This Decision tree algorithm processes input vector X and examines futures to find the optimal method to split the data into subsets.

$$f(X) = \text{DecisionTree}(X)$$

Training Process: It uses criteria such as Gini impurity or information gain (entropy) in order to create the most homogeneous subsets concerning Y. The model splits recursively until it reaches maximum tree depth or minimum number of sample in a leaf.

$$\text{Gini Impurity} \to Gini(D) = 1 - \sum_{i=1}^{C} p_i^2$$

$$\text{Information Gain (Entropy)} \to H(D) = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

- C: number of classes in the dataset
- p_i: proportion of samples belonging to class i in the dataset D

Information Gain Calculation computes the reduction in entropy after splitting dataset.

$$IG(D, A) = H(D) - \sum_{v \in \text{Value}(A)} \frac{|D_v|}{|D|} H(D_v)$$

- D_v: subset of dataset
- $|D_v|$: number of samples in subset D_v
- $|D|$"total number of samples in D

For the new input, the model traverses from the root node down to a leaf node based on the feature values in X and at each node, it applies a decision rule based on the value of features (For example, checking if the `baseline_value` is greater than a certain threshold) to decide whether to move left or right in the tree. Once a leaf node is reached, the predicted class label associated with that leaf is returned as the output $f(X)$.

4.1.2 Mathematical Formulation for Random Forest

The Random Forest algorithm trains multiple decision trees. Random subsets of the dataset and features are selected for training each tree in order to reduce over-fitting. Each tree is trained on these subsets using the same mechanisms, evaluating the input features to make splits based on impurity measures like Gini impurity or information gain. The final Random Forest model comprises T individual decision trees which trained independently.

For a new input X, trees (T) in the Random Forest makes an independent prediction it means each tree returns its predicted class label. Then, Random forest aggregates these predictions and the final predicted class is determined using majority voting.

$$f(X) = \text{majority vote}(\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_T)$$

4.1.3 Mathematical Formulation for k-Nearest Neighbors (KNN)

The k-Nearest Neighbors (KNN) algorithm classifies a fetal health instance based on the proximity to its neighbors in the feature space. For a given input vector X which represents clinical features such as fetal heart rate and other monitoring metrics, KNN identifies the k closest instances in the Fetal Health Database and assigns the class label based on the majority vote from these neighbors.

$$f(X) = \text{KNN}(X, k)$$

Distance Calculation: KNN uses a distance metric to find the closest neighbors. For the Fetal Health Database, the most commonly used distance metric is Euclidean distance:

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

- p: input instance representing fetal clinical features
- q: training instance from the Fetal Health Database
- n: number of features (e.g., baseline fetal heart rate, accelerations, etc.)

Classification Process: 1. For a new fetal health input X, the algorithm computes the distance to all instances in the Fetal Health Database. 2. It selects the k nearest neighbors based on the calculated distances. 3. The predicted health status for X is determined by a majority vote of the k neighbors' class labels, where the classes could represent healthy, suspicious, or abnormal fetal conditions.

$$f(X) = \text{mode}(y_1, y_2, \ldots, y_k)$$

- y_i: health status class label of the i-th nearest neighbor

Choosing k: The choice of k is crucial in KNN and should be determined through cross-validation. A lower k might lead to a model that captures noise, while a higher k smooths the decision boundary, which may overlook significant patterns in fetal health. **Weighting Neighbors:** In certain implementations, neighbors can be weighted according to their distances. Closer neighbors have a more significant influence on the classification outcome.

$$f(X) = \frac{\sum_{i=1}^{k} w_i y_i}{\sum_{i=1}^{k} w_i}$$

- $w_i = \frac{1}{d(X, q_i)}$: weight assigned inversely proportional to the distance

For the new input, the distances to each instance in the Fetal Health Database are computed, and the model predicts the fetal health status based on the majority class among the k nearest neighbors.

4.2 Mathematical behind Evaluation Metrics

In this project, The confusion matrix is considered for models evaluation based on comparison of true classes with predicted classes which has been done by models on test dataset (unseen data). For the multi-class classification problem with classes $C = \{0, 1, 2\}$ (representing Normal, Suspicious, and Pathological), t is represented as:

Actual\Predicted	0	1	2
0	TP_{00}	FP_{01}	FP_{02}
1	FN_{10}	TP_{11}	FP_{12}
2	FN_{20}	FN_{21}	TP_{22}

Where:

- TP_{kk}: True Positives for class k (correct predictions).
- FP_{kj}: False Positives; predicted class k when the actual class is j.
- FN_{jk}: False Negatives; actual class j predicted as k.

4.2.1 Key Metrics

1. Accuracy: rate of total correct samples to the total sample.

$$\frac{\sum_{k=0}^{2} TP_{kk}}{\text{Total samples}} = \frac{TP_{00} + TP_{11} + TP_{22}}{TP_{00} + TP_{01} + TP_{02} + TP_{10} + TP_{11} + TP_{12} + TP_{20} + TP_{21} + TP_{22}}$$

2. Precision: rate of true positive predictions to the total number of positive predictions made by the model

$$\text{Precision} = \frac{TP_{kk}}{TP_{kk} + \sum_{j\neq k} FP_{jk}}$$

3. Recall: rate of the number of true positive (TP) instances to the sum of true positive and false negative (FN) instances.

$$\text{Recall} = \frac{TP_{kk}}{TP_{kk} + \sum_{j\neq k} FN_{jk}}$$

4. F1 Score: It is used to evaluate the overall performance of a classification model. It is the harmonic mean of precision and recall,

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Example of Precision and Recall for Each Class

For the classes Normal (0), Suspicious (1), and Pathological (2):

- Class 0:

$$\text{Precision}_0 = \frac{TP_{00}}{TP_{00} + FP_{00}}$$

$$\text{Recall}_0 = \frac{TP_{00}}{TP_{00} + FN_{00}}$$

- Class 1:

$$\text{Precision}_1 = \frac{TP_{11}}{TP_{11} + FP_{01}}$$

$$\text{Recall}_1 = \frac{TP_{11}}{TP_{11} + FN_{10}}$$

- Class 2:

$$\text{Precision}_2 = \frac{TP_{22}}{TP_{22} + FP_{12}}$$

$$\text{Recall}_2 = \frac{TP_{22}}{TP_{22} + FN_{20}}$$

4.3 Model Training

In order to experiment different classifiers, we use RF, DT and KNN algorithms for predicting fetal health.

4.3.1 Train-test split strategy

Data is Splited into 80% for training and 20% for testing.

```
X_train_pca, X_test_pca, y_train, y_test = train_test_split(x_pca, y, test_size=0.2,
    random_state=42)
```

4.3.2 Decision Tree Implementation

```
model = DecisionTreeClassifier()
model.fit(X_train_pca,y_train )
pred = model.predict(X_test_pca)
print("\naccuracy score:%f"%(accuracy_score(y_test,pred)*100))
```

output: accuracy score: 95.070423

4.3.3 Random Forest Implementation

```
model = RandomForestClassifier()
model.fit(X_train_pca,y_train)
pred = model.predict(X_test_pca)
print("\naccuracy score:%f"%(accuracy_score(y_test,pred)*100))
```

output: accuracy score: 98.826291

4.3.4 KNN Implementation

```
model = KNeighborsClassifier()
model.fit(X_train_pca,y_train)
pred = model.predict(X_test_pca)
print("\naccuracy score:%f"% accuracy_score(y_test,pred)*100))
```

output: accuracy score: 96.244131

4.4 Hyperparameter Tuning

4.4.1 Why in the PCA, we reduce dataset to 14 components?

```
plt.plot(np.cumsum(pca.explained_variance_ratio_))
```

We realized that the number of dimension can be reduced from 22 to 14 while preserving 95% of its variance. Hence the compressed dataset is of 60 percentage of its original size. Regarding to the below plot of the explained variance as a function of the number of principal components , **we observe an elbow in the curve**. The optimal number of principal components is reached when the cumulative variance stops growing fast.

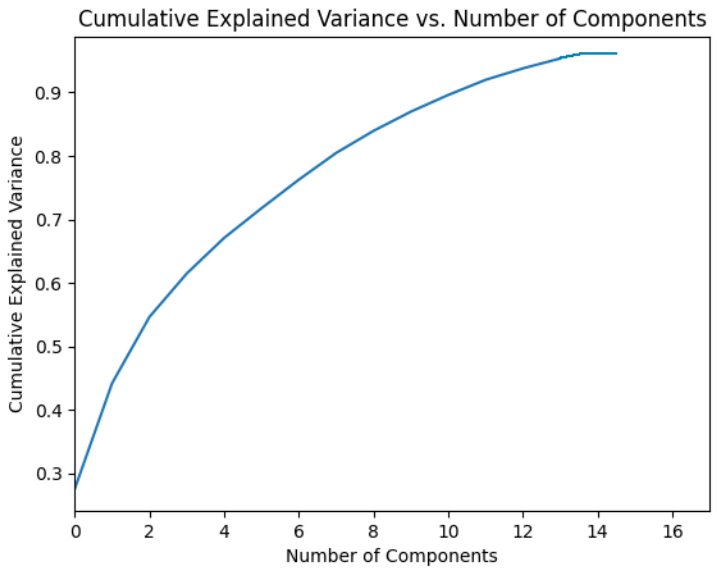


Figure 20: elbow in the curve

In this project, hyperparameter tuning for the classifiers was not conducted, and the default values for all classifiers were used for several reasons:

- **Baseline Performance Assessment:** Utilizing default parameters allows for establishing a baseline performance metric for each model. This initial evaluation helps in understanding how well the classifiers perform with standard settings, which can be useful for future reference if tuning is pursued.

- **Focus on Model Comparison**: The primary aim of this project was to compare various classifiers in terms of their effectiveness in predicting fetal distress. By using default parameters, the emphasis remained on analyzing the differences in performance across models rather than optimizing individual classifiers.

- Complexity of the Task: The fetal health dataset, while informative, may not be large enough to necessitate extensive tuning. In some cases, default parameters yield satisfactory results, especially in initial experiments. It allows for quick identification of promising models without over-fitting or extensive pre-processing.

4.5 Tools and Libraries

In this project, libraries are used as below:

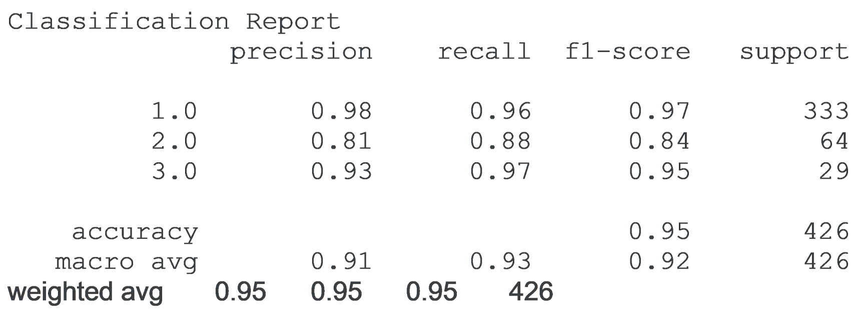
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
#Splitting the data
from sklearn.model_selection import train_test_split
# Algorithms
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
# For checking Model Performance
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import warnings
warnings.simplefilter(action="ignore")
```

5 Results and Discussion

5.1 Models Performance

In this project, we evaluate the model's performance providing the classification report and confusion matrix.

5.1.1 Classification report for Decision Tree



```
Classification Report
              precision    recall  f1-score   support

         1.0       0.98      0.96      0.97       333
         2.0       0.81      0.88      0.84        64
         3.0       0.93      0.97      0.95        29

    accuracy                           0.95       426
   macro avg       0.91      0.93      0.92       426
weighted avg       0.95      0.95      0.95       426
```

Figure 21: Classification report for Decision Tree

5.1.2 Classification report for Random Forest

```python
print("Classification Report")
print(classification_report(y_test, pred))
```

```
Classification Report
              precision    recall  f1-score   support

         1.0       0.99      1.00      0.99       333
         2.0       1.00      0.92      0.96        64
         3.0       1.00      1.00      1.00        29

    accuracy                           0.99       426
   macro avg       1.00      0.97      0.98       426
weighted avg       0.99      0.99      0.99       426
```

Figure 22: Classification report for Random Forest

5.1.3 Classification report for KNN

```python
print("Classification Report")
print(classification_report(y_test, pred))
```

output:

```
Classification Report
              precision    recall  f1-score   support

         1.0       0.97      1.00      0.98       333
         2.0       0.93      0.81      0.87        64
         3.0       0.96      0.90      0.93        29

    accuracy                           0.96       426
   macro avg       0.95      0.90      0.93       426
weighted avg    0.96    0.96    0.96    426
```

Figure 23: Classification report for KNN

5.2 Visual Representation

5.2.1 Confusion Matrix for Decision Tree

```python
ax = plt.subplot()
sns.heatmap(confusion_matrix(y_test, pred), annot=True, ax = ax, cmap = "summer")

# labels, title and ticks
ax.set_xlabel("Predicted labels")
ax.set_ylabel("True labels")
ax.set_title("Confusion Matrix")
ax.xaxis.set_ticklabels(["Normal", "Suspect", "Pathological"])
```

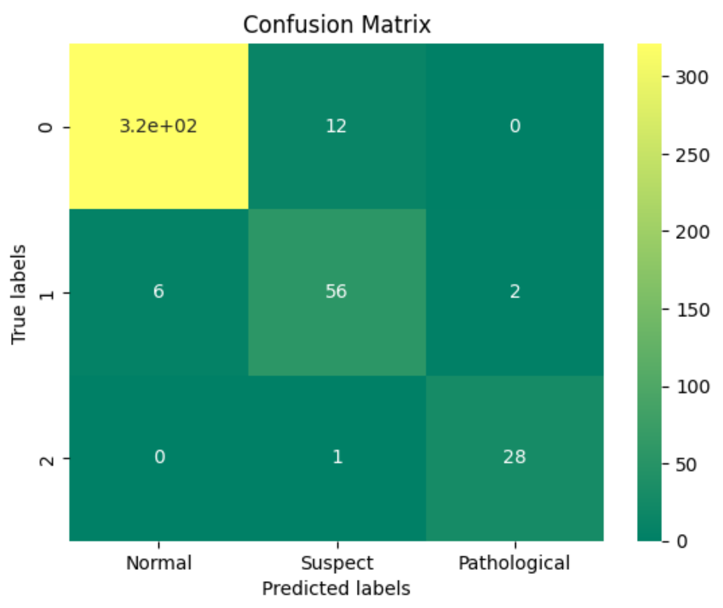


Figure 24: Confusion Matrix for Decision Tree

5.2.2 Confusion Matrix for Random Fores

```python
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test, pred), annot=True, ax = ax, cmap = "summer");

# labels, title and ticks
ax.set_xlabel("Predicted labels")
ax.set_ylabel("True labels")
ax.set_title("Confusion Matrix")
ax.xaxis.set_ticklabels(["Normal", "Suspect", "Pathological"])
```

5.2.3 Confusion Matrix for KNN

```python
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test, pred), annot=True, ax = ax, cmap = "summer");

# labels, title and ticks
ax.set_xlabel("Predicted labels")
ax.set_ylabel("True labels")
ax.set_title("Confusion Matrix")
ax.xaxis.set_ticklabels(["Normal", "Suspect", "Pathological"])
```

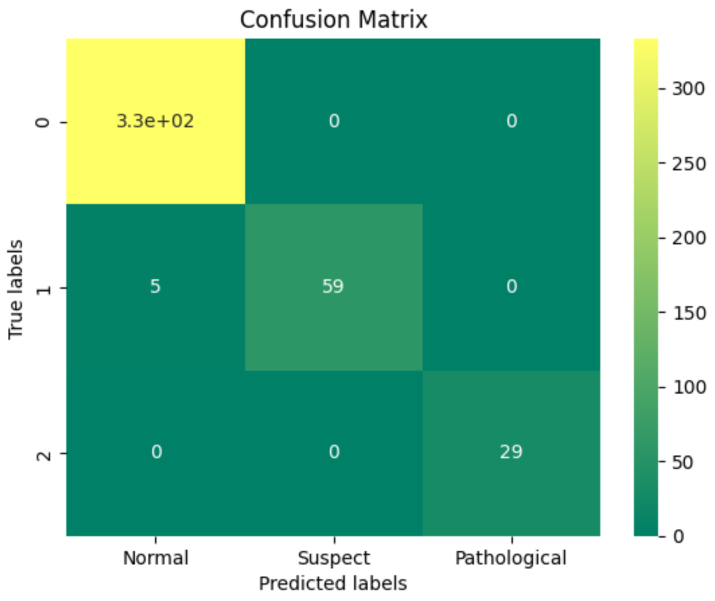


Figure 25: Confusion Matrix for Random Fores

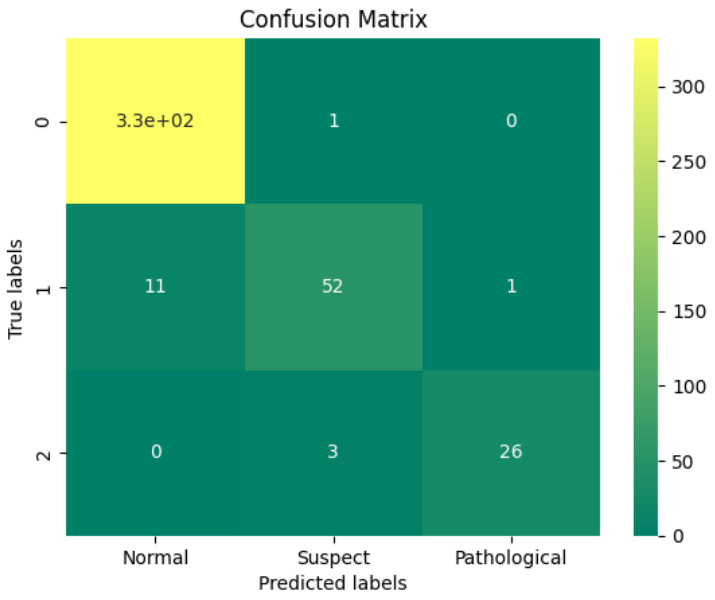


Figure 26: Confusion Matrix for KNN

5.3 Interpretation

5.3.1 Analyze the DR based on report and confusion matrix:

- Precision: This high value for Class 1 means this model is very accurate predicting this class. Class 2 has lower score which means it may produce more false positives.
- Recall: For Class 1 recall is high, shows most actual positives are correctly identified. Class 2 has a little lower value, means some positive instances may be missed.

- F1-Score: This value suggests strong performance across the classes, with Class 2.0 which is the weakest.

Overal: The decision tree classifier works well in this data, especially for Class 1, with high precision, recall, and F1 scores. Class 2 has needs some improvement because of lower precision and recall. The overall accuracy indicates the model is effective for the dataset, but it is not balanced performance across all classes.

5.3.2 Analyze the RF model based on report and confusion matrix:

- **Precision:** These high values shows that this model is good at avoiding false positives.
- **Recall:** These high values means this model successfully identifies most of the actual positive instances.
- **F1-Score:** These high F1-scores means performance is high across all classes.

The overall accuracy of the model is 0.99. The classification report indicates that the model performs well, in particular for Class 1 and Class 3 because of very high precision, recall, and F1-scores. The performance for Class 2 is a little lower but still strong so the model is working well in classifying fetal health outcomes which can be seen in the confusion matrix.

5.3.3 Analyze the KNN based on report and confusion matrix:

- **Precision:** This model depicts very high precision for all of the classes which means when it predicts a class, it is probably correct.
- **Recall:** Class 1 shows a great recall and it means all actual instances in this class are identified correctly. Class 2 shows lower score in recall which means some instances are not captured. For Class 3 some true instances are missed.
- **F1-Score:** This value for class 1 is strong due to its high precision and recall. Class 2 F1-score is lower because of its recall decrease. Class 3 performs well, by high precision and good recall.

Overall: This model performs well, in particular for Class 1 with high precision, recall, and F1-score. Class 2 shows a need for improvement, especially in recall, meaning more tuning is needed to capture all relevant instances.

5.4 Comparison of Models

```python
models = []
models.append(('RF', RandomForestClassifier()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('DT', DecisionTreeClassifier()))
names = []
scores = []
for name, model in models:
    model.fit(X_train_pca, y_train)
    y_pred = model.predict(X_test_pca)
    scores.append(accuracy_score(y_test, y_pred))
    names.append(name)
tr_split = pd.DataFrame({'Name': names, 'Score': scores})
print(tr_split)
```

```
Name      Score
0    RF  0.976526
1   KNN  0.962441
2    DT  0.946009
```

```python
import seaborn as sns
axis = sns.barplot(x = 'Name', y = 'Score', data =tr_split )
axis.set(xlabel='Classifier', ylabel='Accuracy')
for p in axis.patches:
    height = p.get_height()
    axis.text(p.get_x() + p.get_width()/2, height + 0.005, '{:1.4f}'.format(height),
    ha="center")
plt.show()
```

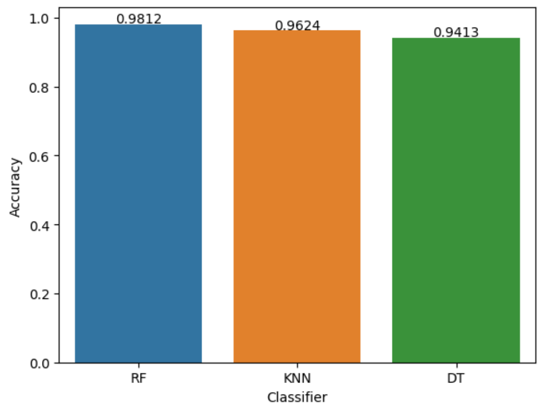


Figure 27: Comparison of Models

The above comparison shows the accuracy scores of classification algorithms. Clearly it can be concluded that **Random Forest** is an optimal model of choice of the given dataset as it has relatively the highest combination of accuracy followed by **K-Nearest Neighbors** and **Decision tree**.

6 Conclusion and Future Work

6.1 Summary of Findings

- This project assessed the effectiveness of three machine learning algorithms including Decision Tree, Random Forest, and k-Nearest Neighbors—in classifying fetal health status based on a dataset retrieved from Kaggle.
- The dataset went through preprocessing, including scaling, and dimensionality reduction (PCA).
- The results showed that the Random Forest model achieved the highest accuracy (approximately 98%), followed closely by k-Nearest Neighbors and Decision Tree.

6.2 Key takeaways from the project

- Random Forest demonstrates better performance compared to Decision Tree and k-Nearest Neighbors for this specific fetal health dataset.
- PCA reduced the dimensionality of the data without significant information loss, improving model efficiency and potentially mitigating overfitting.
- The classification performance highlights the potential of machine learning for early fetal distress detection.

6.3 Conclusion

- The findings suggest that machine learning, specifically Random Forest, offers an approach to predict fetal health classification.
- The high accuracy achieved by the Random Forest model indicates its potential for assisting healthcare professionals in early detection of fetal distress.
- However, the model's performance should be further validated on a larger dataset that is representative of different populations and clinical settings.

6.4 Limitations

- The study was based on a single dataset, which may not fully capture the diversity of fetal health conditions and might limit the generalizability of the findings.

- The interpretation of the model's predictions may require expert clinical knowledge, as it relies on multiple features and complex relationships between them.

6.5 Future Work

- Validate the models on larger, independent datasets to confirm their generalizability.
- Conduct comprehensive hyperparameter tuning to potentially further improve model performance and investigate the impact of parameter choices.
- Explore more advanced machine learning techniques like deep learning or ensemble methods.
- Investigate the interpretability of the models to understand the decision-making process better and provide insights into the factors contributing to fetal health outcomes.

References:

1. Cleveland Clinic Organization.Available online

2. World Health Organization.Available online

3. American Pregnancy Association.Available online

4. Kaggle.Available online