CLASSIFICATION OF MATERNAL HEALTH RISKS USING BOOSTING TECHNIQUE

[1]Caleb Adithia Kurniawan, [2]Rosita Herawati"
[1,2]Program Studi Teknik Informatika Fakultas Ilmu Komputer, Universitas Katolik Soegijapranata
[1]21k10021@student.unika.ac.id, [2]rosita@unika.ac.id

ABSTRACT (ABSTRACT TITLE)

Maternal health is an important concern as it directly impacts the ongoing survival and well-being of future generations. In many developing nations, maternal mortality has become a serious problem despite the advances in medical science. Recognizing potential risks in pregnancies is essential for the well-being of the mother and the newborn. In various clinical applications, including disease diagnosis, treatment planning, and patient monitoring, AI models are capable of showing promising results. Depending on the chance of complications during pregnancy, it can be categorized into three risk levels: low, and moderate. The risk is classified based on age, systolic and diastolic blood pressure, blood sugar, body temperature, and heart rate. This paper aims to apply several boosting techniques: Boosted Random Forest, XGboost, and Catboost to classify the maternal health risk. By classifying the maternal risk, it should help minimize the occurrence of maternal death which will lead to the continuation of humanity.

Keywords: classification, boosting, machine learning

INTRODUCTION

Maternal health plays a critical role as it directly impacts the ongoing survival and well-being of future generations. In many developing nations, maternal mortality has become a serious problem despite the advances in medical science [1], [2] .In 2020, almost 800 pregnant women died from preventable causes every day, and 95% of maternal deaths occurred in low and lower-middle-income countries [3]. Maternal mortality is defined as the death of a woman due to complications during pregnancy or within 42 days after giving birth, regardless of the location or stage of pregnancy [2]. These issues are usually made worse by the handling of pregnancy rather than accidents [2].

Recognizing potential risks in pregnancies is essential for the well-being of the mother and the newborn. However, this task can be challenging due to the fact that the factors involved are very complex and varied. Depending on the chance of complications during pregnancy, it can be categorized into three risk levels: low, moderate, and high [4]. Nowadays, artificial intelligence (AI) has become a method to improve medical science that simplifies analyzing and interpreting complex medical data. In various clinical applications, including disease diagnosis, treatment planning, and patient monitoring, AI models are capable of showing promising results [5]. Ensemble learning methods are one of the machine learning methods that is often used for medical diagnosis purposes [6]. A study in 2023 by Togunwa et al. compared 16 algorithms to classify the risk level of pregnancy. Random Forest Classifier has the highest accuracy when compared to

other single algorithms. Furthermore, a hybrid model from the ANN (artificial neural network) and the RF (random forest) classifier shows the highest accuracy at 94.9% compared to the rest of the 15 algorithms [5].

A study in 2023 by Banujan et al. used a boosting ensemble method for COVID-19 death prediction with models of XGBoost, CatBoost, and LightBGM that attained an accuracy of 98%, AdaBoost attained 96%, and ANN achieved 93%. It indicates that boosting techniques can be more accurate than ANN [7]. So I suggest using an ensemble method using a boosting technique.

LITERATURE STUDY

Togunwa et al. [5] conducted a research study about maternal health risk classification in pregnancy using machine learning. The dataset used in this research was obtained from the Kaggle website which consists of maternal health care. It was used to find whether a hybrid model of machine learning and deep learning can classify maternal health risks and have the highest accuracy after comparing it with other models. This research focuses on the potential of AI to classify maternal health risks. This research is useful to my research as we use the same topic and dataset. The gap in this article is that the authors didn't compare the models with models of a machine learning algorithm with a boosting algorithm; thus, the hybrid method had the highest results in precision, recall, F1 score, and accuracy. This research will be used as the basis of my research for the methodology.

In research conducted by Iwendi et al. [8] about the health prediction of COVID-19 patients. The dataset was also used from the Kaggle website, which compiled it from various sources to make AI models for predicting patient health. This research focuses on making a quick and efficient prediction of a patient using AI techniques. This article is useful as it proposes the same method as my research, which is to use a boosted random forest model to predict a medical condition. The gap in this research is that the authors didn't compare which boosting algorithm has higher accuracy in a boosted random forest model. This research indicates that the boosted random forest algorithm has the capability to provide accurate predictions even when dealing with imbalanced datasets. This research is proof that using a boosted random forest in the medical field is possible.

With the same concept of algorithm, Mishina et al. [9] compare a boosted random forest and a random forest only. This research used five data sets, Pendigits, Letter, Satellite, Spam Base, and Iris, from the UCI Machine Learning Repository to train and evaluate these two models. It compares the computational cost and memory usage between the two models. The main limitation of this research is that they didn't give information about what boosting algorithm they used in the research. The results of this research show that random forests require a total memory of 47% more than boosted random forests. This research was only used for additional informational purposes.

Another research conducted by Banujan et al. [7] is also about the health prediction of COVID-19 patients. The dataset was obtained from the Kaggle website, which contains blood samples from 4313 COVID-19 patients, to identify the most effective prediction model. This

research focuses on finding an effective model by comparing boosting ensemble algorithms and artificial neural networks. This research is useful to my study as it has information about boosting algorithms that can do better than ANN. The limitation of this study is that this model might not be able to predict with another institute dataset or have a lower level of accuracy. This research results in accuracy indicate that an ensemble technique of boosting has a higher accuracy than using ANN. This research will be used as supplementary information for my research in choosing which boosting algorithm will be used.

Furthermore, research by Sanjaya et al. [10] on predicting bad credit problems in a bank. This research uses data from one of the banks in Indonesia to predict bank loan defaults. This research focuses on comparing deep neural networks, random forests, and boosted random forests that consist of adaptive boosting and random forests. This research is useful to my research as not every dataset that is used to make a boosted random forest model can give a better result than a random forest alone. The gap in this research is that there are still some boosting algorithms that can be used to compare. This research indicates that with this dataset, deep neural networks have the highest accuracy compared to the rest of the model. This research will be useful as an information source on boosted random forests.

Abuelezz et al. [11] review the contribution of AI to pregnancy. The authors use a total of 1,753 articles or studies from Google Scholar and PubMed on the use of AI in pregnancy to identify which type of AI is commonly used in the pregnancy field. This review aims to explore features of technologies such as AI for pregnant women. The article is useful to my research topic as it gives information about the use of AI in the pregnancy field. The limitation of this review is that they don't find studies that discuss the effectiveness of models for treating disorders and predicting and diagnosing pregnancies. This review suggests that AI technologies have the potential to improve the healthcare services of pregnant patients. This review is proof that AI can help improve healthcare for treating pregnant patients.

Fauzi et al. [12] conducted research with a boosted random forest for predicting a DDoS attack. It uses a dataset called CICDDoS_2019 to find a solution to predict a DDoS attack. It focuses on comparing random forest and boosted random forest that consists of random forest and ADA boost to detect a DDoS attack. This research is useful as it gives an idea of how to make a model of a random forest with a boosting algorithm. The gap in this research is that it only uses one type of boosting algorithm, and has an accuracy of 99% for both models that can lead to overfitting. This research can be used for the method of making boosted random forests.

Another research by Srivardhan [13] also uses a boosted random forest that consists of ADA boost and random forest to interpret well logs and determine reservoir properties. There are 5 wells that were used for the dataset. This research is useful to learn how to implement a boosted random forest. The gap in this research is that it only uses one type of boosting algorithm. This research result indicates that when the random forest is combined with adaptive boosting, it improves the performance. The random forest model seems to have overfitted, with the training and testing accuracy of 98.21% and 77.62%. The boosted random forest shows a better test accuracy of

97.03% and a training accuracy of 99.40%. This research is proof that using adaptive boosting can improve the random forest's accuracy. So the research method for making a boosted random forest model can be used for my research.

A survey by Mienye and Sun [6] regarding the concept and application of the ensemble method. For machine learning researchers who want to comprehend ensemble learning and the widely used ensemble algorithms, this paper will be an essential reading. The three primary types of ensemble methods that are covered in this work are stacking, boosting, and bagging. This survey found that ensemble learning algorithms have been widely used in a variety of classification and regression tasks across a wide range of disciplines, including medical diagnosis, fraud detection, sentiment analysis, and anomaly detection, because of their robust learning capabilities.

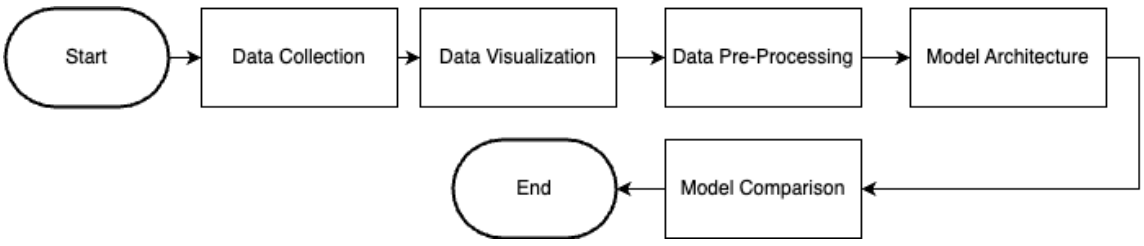
A review and analysis by Ahmed et al. [14] about the risk factors of maternal health using the Internet of Things. This research uses machine learning algorithms to discover the risk level in pregnancy. This research is useful as the dataset that will be used in our research. It explains the correlation between the parameters that will be used to predict the risk in the dataset with the help of a medical expert. It uses a logistic model tree to classify and predict the risk level. It resulted in a 90% accuracy of prediction. This research will help to make sure that the risk in the dataset is based on medical expert opinion.

The research conducted by Togunwa et al. [5] didn't use a boosted random forest, CatBoost algorithm, and XGBoost algorithm. So this research will be conducted with the same preprocessing method as the research conducted by Togunwa et al. [5] to find the precision, recall, F1 score, and accuracy, but using a different set of algorithms. Boosted random forest models that use a combination of random forest classifier and AdaBoost algorithm, CatBoost algorithm, and XGBoost algorithm. Then, it will be compared to see which algorithm has the highest accuracy.

RESEARCH METHODOLOGY

Methodology Flowchart

This flowchart consists of the research implementation steps. The implementation of this research will start by collecting the dataset from the Kaggle.com website. Then the dataset will be visualized to see the statistics of the dataset and providing guidance for the preprocessing steps.



Gambar 1. Methodology Flowchart

Then we preprocess the dataset based on the information that we found at the dataset visualization. The next step is training and testing the model to get the classification results. Then all models will be compared to create a conclusion of this research.

Dataset Collection

The dataset was taken from the Kaggle.com site with the title of Maternal Health Risk Data. The University of California (UCI) machine learning repository provided the maternal health risk dataset used in this investigation. Using an Internet of Things risk monitoring system, the data was first gathered from several hospitals, community clinics, and maternal health care centers located in Bangladesh's rural districts [5]. The dataset uses a CSV format file that consists of 1,014 instances in the dataset that are divided into three categories based on the risk class. 406 instances in the low-risk class, 336 instances in the mid-risk class, and 272 instances in the high-risk class. The dataset consists of age, systolic and diastolic blood pressure, blood sugar, body temperature, heart rate, and risk level. The dataset can be found via the link https://www.kaggle.com/datasets/csafrit2/maternal-health-risk-data.

Dataset Visualization

Data visualization will help researchers to know what needs to be done when doing the dataset pre-processing. First, the data distribution of data targets or the risk level of maternal health risk will be visualized to check whether the data is balanced or not. Second, the data training statistic summary will be visualized to check mean, standard deviation, minimum, and maximum value. The summary is useful for understanding data distribution and detecting outliers.

Data Pre-Processing

This preprocessing uses the same way from Togunwa et al. research [5]. It will be done with 3 steps, namely :

1. Check if there are any missing values from the dataset. Missing data can introduce bias in model training, decreasing accuracy and making the model less reliable.
2. Encode the target categorical variables into numerical variables to facilitate computation. Low-risk, mid-risk, and high-risk classes were coded as 0, 1, and 2. Encoding helps the model to differentiate classes in a structured manner and make more accurate predictions.
3. Check if there are any outliers. Then re-imputed the outliers with the mode value. Outliers are extreme values that can interfere with model training because they can affect the distribution of the data and cause the model to learn unusual patterns or noise. By identifying and dealing with outliers, we can reduce the negative impact of unusual data. Replacing outliers with mode values helps maintain the integrity of the original data and ensures that unusual values do not distort model results.

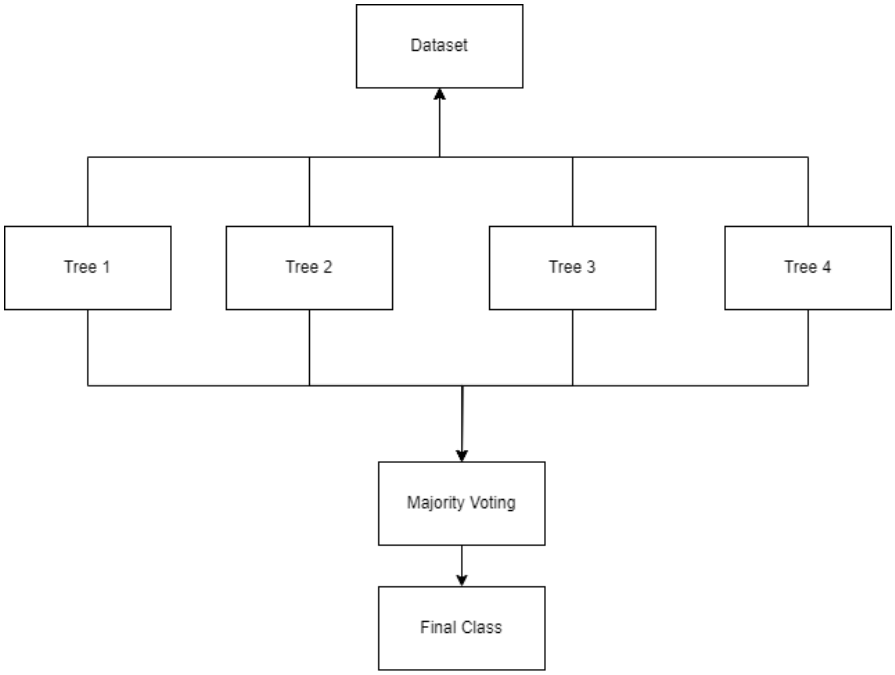
Dataset Splitting

The dataset will be split into data training and data testing. Respectively, the splitting will be 60%-40%, 70-30%, and 80%-20%. Furthermore, to avoid overfitting, this research will use 10-fold cross validation. 10-fold cross validation means the training dataset is divided into 10 equal parts so it provides a more accurate and stable estimate than single train-test splits by balancing both bias and variance.

Model Architecture

After the preprocessing is done, the next step is to build/train the model of boosted random forest that uses a combination of random forest classifier with Ada boost algorithm and boosting model that uses Cat Boost and XG Boost. After the training is done, we will test the model for overfitting assessment.

Random forest is one of the ensemble algorithms. It uses the bagging technique to build multiple decision trees using bootstrapped samples. It solves the overfitting problem that often happens in decision trees. The steps of the random forest classifier can be seen in Gambar 2.

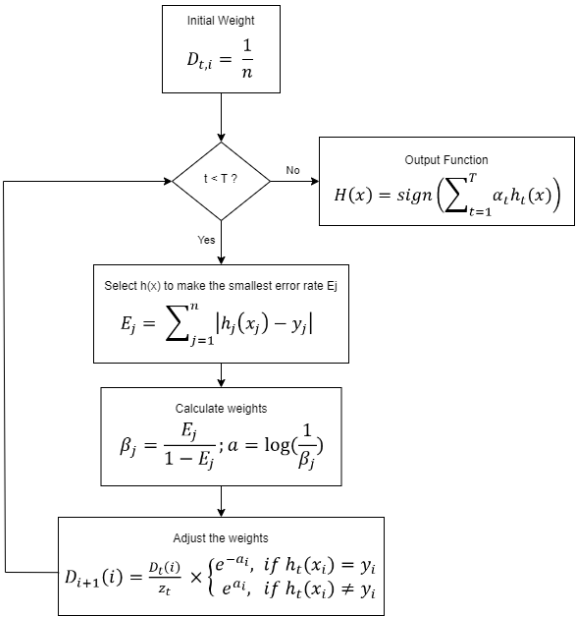


Gambar 2. Random Forest Classfier

Random Forest creates multiple subsets of the training data through bootstrap sampling. It means that, by sampling with a replacement, it creates several random subsets of the data, each size N, from the original dataset with N samples. Some samples may appear multiple times in the same subset or not appear at all. For each bootstrap sample, a decision tree is built. To add variety and randomization, random forests use two main methods. The first one is bootstrap sampling.

Each tree in the forest is trained on a different bootstrap sample, which is a subset of the training data generated by sampling with replacement. This means that each bootstrap sample will have some repeated samples and may leave out others, adding diversity to the trees. The second one is random feature selection. For Each node in the tree, only a random subset of features is considered for splitting. This feature prevents the trees from relying too heavily on the most predictive features and further promotes variation among the trees in the forest.

The first boosting algorithm that will be used is AdaBoost because this method has been used by Iwendi et al. [8] in their research on boosted random forest for the health prediction of COVID-19 patients. Adaptive Boost can be used to obtain a robust classifier using weak learners. The first step in this algorithm is typically using the basic classification algorithm. Then adjust the sample weight according to the base classification result, which makes the classified sample more noticeable. Subsequently, the custom sample is used to train the next base learner. After iteration, weighing is added to the base learner to form the final classification. It can be seen in Gambar 3 from Sanjaya et al. research [10].



Gambar 3. Ada Boost Classifier

To combine random forest classifier and adaptive boosting, we will use Random Forest as a base learner for AdaBoost, which means each decision tree model in Random Forest is considered a "weak learner" by AdaBoost. We use AdaBoost to combine multiple Random Forest models into one more powerful model. AdaBoost sequentially strengthens model performance by adjusting the weights for each data sample based on previous model performance, making it possible to focus attention on samples that are difficult to predict.

The second boosting algorithm is XGBoost. It is an implementation of gradient boosting. The model is built by combining several decision trees as the base model with weak learners to

produce a stronger prediction. Every new decision tree model is built based on the gradient and hessian of the previous tree. XGBoost loss function contains a regularization term $\Omega(h_m)$ from function (3) where γ represents the complexity parameter, T represents the number of leaves in the tree, λ is a penalty parameter, and ω denotes the output of the leaf nodes. It prevents overfitting that optimizes the objective function which $F(x_i)$ represents the prediction on the $i-th$ instance at the $M-th$ iteration. L represents a loss function that computes the differences between the predicted class and the actual class of the target variable that is represented in function (1).

$$L_M\big(F(x_i)\big) = \sum_{i=1}^{n} L\big(y_i, F(x_i)\big) + + \sum_{m=1}^{M} \Omega(h_m) \tag{1}$$

$$\Omega(\mathrm{h}) = \gamma \mathrm{T} + \frac{1}{2}\,\lambda \|\omega\|^2 \tag{2}$$

While gradient boosting uses the first derivative of Taylor approximation, XGBoost uses the second derivative that can be seen in function (3). The final loss value is determined by summing the loss values of all leaf nodes, assuming that I_j represents the samples in leaf node j.

$$L_M = \left[\left(\sum_{i\in I_j} g_i\right)\omega_j + \frac{1}{2}\left(\sum_{i\in I_j} h_i + \lambda\right)\omega_j^2\right] + \gamma \mathrm{T} \tag{3}$$

$$g_i = \frac{\partial \mathrm{l}\big(y_i, F(x_i)\big)}{\partial y_i} \tag{4}$$

$$h_i = \frac{\partial^2 L\big(y_i, F(x_i)\big)}{\partial y_i^2} \tag{5}$$

The third one is CatBoost because in a reseach by Banujan et al. [7] it have the highest accuracy for the health prediction of COVID-19 patients after comparing it with other boosting algorithm. CatBoost algorithm is an implementation of gradient boosting. It use the same training process as XGBoost as both is an implementation of gradient boosting. CatBoost excludes a number of samples that will not be used to train the current model and estimate the gradient of each sample at every boosting iteration. So it can lowers overfitting by unbiased gradient estimations. During the training stage, the algorithm manages categorical features well. Using a random permutation of the training set, the algorithm determines the average label value for each sample in the permutation that has the same category value as the sample that is provided [6]. If $\sigma = (\sigma 1, \ldots, \sigma_n)$ is the permutation, then $x_{\sigma_p,k}$ is replaced with function (6) where P is a prior value, and a is the weight of the prior value. Meanwhile, the parameter $a > 0$.. By adding the prior value and the prior weight, it reduce the noise of low-frequency categories.

$$\frac{\sum_{j=1}^{p-1}\left[\, x_{\sigma_p,k} = x_{\sigma_j,k} \,\right] Y_{\sigma_j} + a.p}{\sum_{j=1}^{p-1}\left[\, x_{\sigma_p,k} = x_{\sigma_j,k} \,\right] Y_{\sigma_p} + a} \quad (6)$$

Compare The Algorithms

After training the model, we need to test all of the models and compare it by the values of precision, recall, F1 score, and accuracy to see which models have a higher result in classifying maternal health risk. To calculate precision, recall, F1 score, and accuracy, we can see functions (7), (8),(9),(10)

$$Precision = \frac{TP}{TP + FP} \quad (7)$$

$$Recall = \frac{TP}{TP + FN} \quad (8)$$

$$F1\ Recall = \frac{2(Recall * Precision)}{(Recall + Precision)} \quad (9)$$

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \quad (10)$$

IMPLEMENTATION AND RESULTS

Experiment Setups

This research uses Google Collab with a disk of 107.7 GB and Python version 3. Researchers used several libraries to help with the implementation. The Sklearn library was used to help several things. First, creating models for Random Forest Classifier and AdaBoost Classifier. Second, splitting the dataset into specific sizes. Third, calculating accuracy, precision, recall, and F1 scores. Fourth, creating classification reports and confusion matrix. Fifth, to create the K-Fold and cross validating the model. The Xgboost library is used to model XGBoost Classifier. Then the Catboost library to model CatBoost Classifier. Fourth, Pandas library to read data from CSV. Fifth, Seaborn and Matplotlib library to display the confusion matrix.

Implementation

This section will explain the implementation of classification of maternal health risk. It will be done based on the Research Methodology. It will find a suitable splitting and parameters for each method/model. Then the model will be trained and tested to make a conclusion. There will be a process explanation of each part and the code that will be used.

Data Collection

The data collection will be done by importing a CSV file that has been downloaded from the Kaggle website. The file is called "Maternal Health Risk Dataset.csv". The downloaded file is stored inside the researcher's Google Drive folder. Every time the Python file runs for the first time, Google Collab will ask permission to access the Google Drive.

```
1. from google.colab import drive
2. drive.mount('/content/drive')
3. import pandas as pd
4. data = pd.read_csv('/content/drive/My Drive/Maternal Health Risk Data Set.csv')
```

The implementation starts by importing data from Google Drive then reading the csv file using pandas. After that we save the readed csv file into the data variable. It can be seen at lines 1 & 2 is used to connect the colab to researcher personal google drive. Then at line 3 is used to install the pandas library to read the dataset from the drive.

Data Visualization

The data will be visualized into a chart and a report. It will visualize the data distribution and the statistics of the dataset. So it will help what needs to be done for the pre-processing part.

```
5. import matplotlib.pyplot as plt
6.
7. risk_distribution = data['RiskLevel'].value_counts()
8. plt.figure(figsize=(8, 6))
9. risk_distribution.plot(kind='bar', color=['green', 'orange', 'red'])
10.  plt.title('Distribution of Dataset Based on Risk Level')
11.  plt.xlabel('Risk Level')
12.  plt.ylabel('Number of Subjects')
13.  plt.show()
14.
15.  summary = data.drop(columns=['RiskLevel'])
16.  summary_stats = summary.describe()
17.  print(summary_stats)
```

The visualization can be seen in lines 7 until 17. The code will visualize the data distribution of RiskLevel through the dataset. To visualize it into a bar chart, the Matplotlib library will be used. Then a statistical summary will be visualized to check the mean, standard deviation, minimum,and maximum value.

Data Pre-Processing

The preprocess will be done not only based on the data visualization. The preprocess will check the missing values. Then it will encode the risk level into numerical.

```
18.  missing_values = data.isnull().sum()
19.  data['RiskLevel'] = data['RiskLevel'].map({'low risk': 0, 'mid risk': 1, 'high risk': 2})
20.  heart_rate_mode = data['HeartRate'].mode()[0]
21.  data.loc[data['HeartRate'] == 7, 'HeartRate'] = heart_rate_mode
```

Then the process of data processing is conducted by checking the missing values and outliers. Then encoding the target category into numerical. The low-risk, mid-risk and high-risk categories are encoded into 0, 1, and 2 respectively. This pre-processing is based on Togunwa et al. [5] research. Line 18 is used to check the missing values from the dataset. Then we check the heart rate outlier and change it with the mode of heart rate at lines 20 & 21.

Cross Validation Function Helper

Then the function helper for cross validation is created to help with the implementation. cross validation will help the model to be trained by several folds of data. This research will use 10-fold cross validation.

```
22.  from sklearn.model_selection import KFold, cross_validate
23.  import numpy as np
24.
25.  def evaluate_model(model, X, y):
26.      scoring = {
27.          'accuracy': 'accuracy',
28.          'precision': 'precision_weighted',
29.          'recall': 'recall_weighted',
30.          'f1': 'f1_weighted'
31.      }
32.      cv = KFold(n_splits=10, shuffle=True, random_state=42)
33.      scores = cross_validate(model, X, y, cv=cv, scoring=scoring,
   return_train_score=False)
34.      return {metric: np.mean(scores[f'test_{metric}']) for metric in
   scoring}
```

The code above is used to cross validate the model by using 10-fold based on accuracy, precision, recall, and F1 for the scoring. 10-fold means the training dataset is divided into 10 equal parts. The model is trained on 9 folds and tested on the 1 remaining fold and then the results are averaged using the numpy library. Lines 22 & 23 are importing K-Fold and cross_validate to help with creating the K-Fold and cross validate it.

Finding Best Parameters Function Helper

Tabel 1. Model Parameters

Models	Parameters
Boosted Random Forest Classifier	n_estimators_range = [25, 50, 75, 100]
	learning_rate_range = [1, 0.5, 0.1]
	max_depth_range = range(5, 11)
CatBoost Classifier	iterations_range = range(100, 1100, 100)
	learning_rate_range = [1, 0.5, 0.1]
	depth_range = range(5, 11)
XGBoost Classifier	n_estimators_range = range(100, 1100, 100)
	learning_rate_range = [1, 0.5, 0.1]
	max_depth_range = range(5, 11)

These several functions are used to help find the best parameters from each model. So each model has their own function to search for the best parameters. Each model parameter is limited to specific type and range based on Tabel 1

The first parameter is the learning rate. The learning rate controls the contribution of each tree to the final model. A higher learning rate speeds up learning but might lead to overfitting, while a smaller rate results in slower learning but more stable convergence. Then the second one, max depth limits the depth of each decision tree, preventing trees from becoming overly complex. A higher value allows the model to capture more intricate patterns, but it risks overfitting. Then the third one, iteration and n_estimators, defines the number of decision trees. More trees generally lead to better performance but increase computational cost.

This function has an X and y as the parameters of the function which will be used to receive the data for training. By looping the model parameters that will be used to create the model, it will create many combinations of parameters and then compare it one by one for the highest result. Then the first function for searching the best parameters is adaboost_best_params. This function is used to help to find the best parameters for Boosted Random Forest. In this research, Boosted Random Forest is using AdaBoost Classifier but the base model is changed into Random Forest Classifier.

```
35.  from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier
36.
37.  def adaboost_best_params(X, y):
38.      best_score = {'accuracy': 0, 'f1': 0, 'recall': 0, 'precision': 0}
39.      best_params = {}
40.
41.      n_estimators_range = [25, 50, 75, 100]
42.      learning_rate_range = [1, 0.5, 0.1]
43.      max_depth_range = range(5, 11)
44.
45.      for n_estimators in n_estimators_range:
46.        for max_depth in max_depth_range:
47.            for learning_rate in learning_rate_range:
48.              base_model   =   RandomForestClassifier(max_depth=max_depth,
   random_state=42)
49.              model       =       AdaBoostClassifier(estimator=base_model,
   n_estimators=n_estimators,  learning_rate=learning_rate,  random_state=42,
   algorithm="SAMME")
50.              scores = evaluate_model(model, X, y)
51.
52.              print(f"Learning    Rate:    {learning_rate},    max_depth:
   {max_depth}, "
53.                    f"n_estimators: {n_estimators}, Scores: {scores}")
54.
55.              if  all(scores[metric]  >  best_score[metric]  for  metric  in
   best_score):
56.                  best_score = scores
57.                  best_params = {
58.                      'learning_rate': learning_rate,
59.                      'max_depth': max_depth,
60.                      'n_estimators': n_estimators
```

```
61.                     }
62.                     print(f"New Best Params: {best_params}, Best Scores:
   {best_score}")
63.
64.       return best_params, best_score
```

The model parameters can be seen in Lines 41-43 based on Table 4.1. The looping process can be seen at lines 45 until 62. To get the results of each combination, the evaluate_model functions will help to cross validate the model by 10-fold and return the mean results of the model at line 50. If the current loop result is higher than the previous loop, then it will be stored in the variable for the best score and best parameters that have been initiated at lines 38-39. After the looping process is done, line 64 will return the best parameters and best score.

Then the second function for searching the best parameters is catboost_best_params. This function is used to help to find the best parameters for CatBoost Classifier. The base model of CatBoost will be Decision Tree Classifier.

```
65.   !pip install catboost
66.   from catboost import CatBoostClassifier
67.
68.   def catboost_best_params(X, y):
69.       best_score = {'accuracy': 0, 'f1': 0, 'recall': 0, 'precision': 0}
70.       best_params = {}
71.
72.       learning_rate_range = [1, 0.5, 0.1]
73.       depth_range = range(5, 11)
74.       iterations_range = range(100, 1100, 100)
75.
76.       for iterations in iterations_range:
77.           for learning_rate in learning_rate_range:
78.               for depth in depth_range:
79.                   model = CatBoostClassifier(
80.                       learning_rate=learning_rate,
81.                       depth=depth,
82.                       iterations=iterations,
83.                       loss_function='MultiClass',
84.                       silent=True
85.                   )
86.
87.                   scores = evaluate_model(model, X, y)
88.
89.                   print(f"Learning Rate: {learning_rate}, Depth: {depth},
   "
90.                         f"Iterations: {iterations}, Scores: {scores}")
91.
92.                   if all(scores[metric] > best_score[metric] for metric in
   best_score):
93.                       best_score = scores
94.                       best_params = {
95.                           'learning_rate': learning_rate,
96.                           'depth': depth,
97.                           'iterations': iterations
98.                       }
99.                       print(f"New Best Params: {best_params}, Best Scores:
   {best_score}")
```

```
100.
101.     return best_params, best_score
```

The model is created at line 79 which calls the CatBoost Classifier model with its parameter and the model parameters can be seen in lines 72-74 based on Table 4.1. The looping process can be seen at lines 76 until 99. To get the results of each combination, the evaluate_model functions will help to cross validate the model by 10-fold and return the mean results of the model at line 87. If the current loop result is higher than the previous loop, then it will be stored in the variable for the best score and best parameters that have been initiated at lines 69 & 70. After the looping process is done, line 101 will return the best parameters and best score.

Then the third function for searching the best parameters is xgboost_best_params. This function is used to help to find the best parameters for XGBoost Classifier. The base model of XGBoost will be Decision Tree Classifier.

```
102. from xgboost import XGBClassifier
103.
104. def xgboost_best_params(X, y):
105.     best_score = {'accuracy': 0, 'f1': 0, 'recall': 0, 'precision': 0}
106.     best_params = {}
107.
108.     learning_rate_range = [1, 0.5, 0.1]
109.     n_estimators_range = range(100, 1100, 100)
110.     max_depth_range = range(5, 11)
111.
112.     for n_estimators in n_estimators_range:
113.       for learning_rate in learning_rate_range:
114.             for max_depth in max_depth_range:
115.                 model    =    XGBClassifier(learning_rate=learning_rate,
   n_estimators=n_estimators,      max_depth=max_depth,      random_state=42,
   objective='multi:softmax', num_class=3)
116.                 scores = evaluate_model(model, X, y)
117.
118.                 print(f"Learning   Rate:   {learning_rate},   max_depth:
   {max_depth}, "
119.                       f"n_estimators: {n_estimators}, Scores: {scores}")
120.
121.                 if all(scores[metric] > best_score[metric] for metric in
   best_score):
122.                     best_score = scores
123.                     best_params = {
124.                         'learning_rate': learning_rate,
125.                         'max_depth': max_depth,
126.                         'n_estimators': n_estimators
127.                     }
128.                     print(f"New Best Params: {best_params}, Best Scores:
   {best_score}")
129.
130.     return best_params, best_score
```

The model is created at line 115 which calls the XGBoost Classifier model with its parameter and the model parameters can be seen in lines 108-110 based on Table 4.1. The looping process can be seen at lines 112 until 128. To get the results of each combination, the evaluate_model

functions will help to cross validate the model by 10-fold and return the mean results of the model at line 116. If the current loop result is higher than the previous loop, then it will be stored in the variable for the best score and best parameters that have been initiated at lines 105 & 106. After the looping process is done, line 130 will return the best parameters and best score.

Model Evaluation Helper

This function is used to help evaluate the model with the best parameters that have been found with the function above. The evaluation is based on the accuracy, precision, recall, and F1 score. It will use the Sklearn library to calculate the evaluation metrics.

```
131. from sklearn.metrics import accuracy_score, precision_score,
   recall_score, f1_score, classification_report, confusion_matrix
132. import seaborn as sns
133. import matplotlib.pyplot as plt
134.
135. def evaluate_metrics(y_true, y_pred):
136.     accuracy = accuracy_score(y_true, y_pred)
137.     precision = precision_score(y_true, y_pred, average='macro')
138.     recall = recall_score(y_true, y_pred, average='macro')
139.     f1 = f1_score(y_true, y_pred, average='macro')
140.
141.     print(f"Accuracy: {accuracy}")
142.     print(f"Precision: {precision}")
143.     print(f"Recall: {recall}")
144.     print(f"F1 Score: {f1}")
145.     print("")
146.
147.     report = classification_report(y_true, y_pred)
148.     print("Classification Report:")
149.     print(report)
150.     cm = confusion_matrix(y_true, y_pred)
151.     plt.figure(figsize=(4, 3))
152.     sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=True,
   yticklabels=True)
153.     plt.title('Confusion Matrix')
154.     plt.xlabel('Predicted')
155.     plt.ylabel('Actual')
156.     plt.show()
157.
158.     return {"accuracy": accuracy, "precision": precision, "recall":
   recall, "f1": f1}
```

The model evaluation function helper can be seen at lines 135 until 158. After calculating the metrics, it will display classification reports to help evaluate the performance of a classification model. Then it will show the confusion matrix to visualize the model performance based on True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN). After that, the function will return the calculated accuracy, precision, recall, and F1 score.

Splitting 70:30 Experiment

This section is about finding the best parameters and score of each model using 70% data training and 30% data testing. First, the dataset is splitted into 70% data training and 30% data testing. X_train and y_train are used for training and X_test and y_test are used to test the model.

```
159. X = data.drop(columns=['RiskLevel'])
160. y = data['RiskLevel']
161. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
   random_state=42)
162.
163. catboost_best, catboost_score = catboost_best_params(X_train, y_train)
164. print(f"CatBoost Best Params: {catboost_best}, Score: {catboost_score}")
165.
166. catboost_70_30    =    CatBoostClassifier(learning_rate=1,    depth=10,
   iterations=400, loss_function='MultiClass', silent=True)
167. catboost_70_30.fit(X_train, y_train)
168.
169. y_pred_catboost = catboost_70_30.predict(X_test)
170. results['CatBoost_70_30'] = evaluate_metrics(y_test, y_pred_catboost)
171.
172. xgboost_best, xgboost_score = xgboost_best_params(X_train, y_train)
173. print(f"XGBoost Best Params: {xgboost_best}, Score: {xgboost_score}")
174.
175. xgboost_70_30    =    XGBClassifier(learning_rate=0.1,    max_depth=9,
   n_estimators=400, objective='multi:softmax', num_class=3)
176. xgboost_70_30.fit(X_train, y_train)
177.
178. y_pred_xgboost = xgboost_70_30.predict(X_test)
179. results['XGBoost_70_30'] = evaluate_metrics(y_test, y_pred_xgboost)
180.
181. adaboost_best, adaboost_score = adaboost_best_params(X_train, y_train)
182. print(f"AdaBoost Best Params: {adaboost_best}, Score: {adaboost_score}")
183.
184. base_model = RandomForestClassifier(max_depth=5, random_state=42)
185. adaboost_70_30          =          AdaBoostClassifier(learning_rate=0.1,
   estimator=base_model, n_estimators=50, algorithm="SAMME")
186. adaboost_70_30.fit(X_train, y_train)
187.
188. y_pred_adaboost = adaboost_70_30.predict(X_test)
189. print("AdaBoost 70:30 Results:")
190. results['AdaBoost_70_30'] = evaluate_metrics(y_test, y_pred_adaboost)
```

The X_train and y_train variables are used to find the best parameters using catboost_best_params function at line 163, xgboost_best_params function at line 172, and adaboost_best_params function at line 181. Then after the best parameters are found, the parameters are used to train the model with X_train and y_train variables and to the X_test variable. After the result is out, we evaluate the classification for each model at line 170, 179, and 190 using the evaluate_metrics function by inputting the y_test which is the real value from the dataset and the classificate value from model classification.

Splitting 80:20 Experiment

This section is about finding the best parameters and score of each model using 80% data training and 20% data testing. First, the dataset is splitted into 80% data training and 20% data testing. X_train and y_train are used for training and X_test and y_test are used to test the model.

```
191. X = data.drop(columns=['RiskLevel'])
192. y = data['RiskLevel']
193. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
   random_state=42)
194.
195. catboost_best, catboost_score = catboost_best_params(X_train, y_train)
196. print(f"CatBoost Best Params: {catboost_best}, Score: {catboost_score}")
197.
198. catboost_80_20    =    CatBoostClassifier(learning_rate=1,    depth=10,
   iterations=400, loss_function='MultiClass', silent=True)
199. catboost_80_20.fit(X_train, y_train)
200.
201. y_pred_catboost = catboost_80_20.predict(X_test)
202. results['CatBoost_80_20'] = evaluate_metrics(y_test, y_pred_catboost)
203.
204. xgboost_best, xgboost_score = xgboost_best_params(X_train, y_train)
205. print(f"XGBoost Best Params: {xgboost_best}, Score: {xgboost_score}")
206.
207. xgboost_80_20    =    XGBClassifier(learning_rate=0.1,    max_depth=9,
   n_estimators=400, objective='multi:softmax', num_class=3)
208. xgboost_80_20.fit(X_train, y_train)
209.
210. y_pred_xgboost = xgboost_80_20.predict(X_test)
211. results['XGBoost_80_20'] = evaluate_metrics(y_test, y_pred_xgboost)
212.
213. adaboost_best, adaboost_score = adaboost_best_params(X_train, y_train)
214. print(f"AdaBoost Best Params: {adaboost_best}, Score: {adaboost_score}")
215.
216. base_model = RandomForestClassifier(max_depth=5, random_state=42)
217. adaboost_80_20          =          AdaBoostClassifier(learning_rate=0.1,
   estimator=base_model, n_estimators=50, algorithm="SAMME")
218. adaboost_80_20.fit(X_train, y_train)
219. y_pred_adaboost = adaboost_80_20.predict(X_test)
220. print("AdaBoost 80:20 Results:")
221. results['AdaBoost_80_20'] = evaluate_metrics(y_test, y_pred_adaboost)
```

The X_train and y_train variables are used to find the best parameters using catboost_best_params function at line 195, xgboost_best_params function at line 204, and adaboost_best_params function at line 213. Then after the best parameters are found, the parameters are used to train the model with X_train and y_train variables and to the X_test variable. After the result is out, we evaluate the classification for each model at line 202, 211, and 221 using the evaluate_metrics function by inputting the y_test which is the real value from the dataset and the classificate value from model classification.

Splitting 60:40 Experiment

This section is about finding the best parameters and score of each model using 60% data training and 40% data testing. First, the dataset is splitted into 60% data training and 40% data testing. X_train and y_train are used for training and X_test and y_test are used to test the model.

```
222. X = data.drop(columns=['RiskLevel'])
223. y = data['RiskLevel']
224. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
   random_state=42)
225.
226. catboost_best, catboost_score = catboost_best_params(X_train, y_train)
227. print(f"CatBoost Best Params: {catboost_best}, Score: {catboost_score}")
228.
229. catboost_60_40 = CatBoostClassifier(learning_rate=1, depth=10,
   iterations=400, loss_function='MultiClass', silent=True)
230. catboost_60_40.fit(X_train, y_train)
231.
232. y_pred_catboost = catboost_60_40.predict(X_test)
233. results['CatBoost_60_40'] = evaluate_metrics(y_test, y_pred_catboost)
234.
235. xgboost_best, xgboost_score = xgboost_best_params(X_train, y_train)
236. print(f"XGBoost Best Params: {xgboost_best}, Score: {xgboost_score}")
237.
238. xgboost_60_40 = XGBClassifier(learning_rate=0.1, max_depth=9,
   n_estimators=400, objective='multi:softmax', num_class=3)
239. xgboost_60_40.fit(X_train, y_train)
240.
241. y_pred_xgboost = xgboost_60_40.predict(X_test)
242. results['XGBoost_60_40'] = evaluate_metrics(y_test, y_pred_xgboost)
243.
244. adaboost_best, adaboost_score = adaboost_best_params(X_train, y_train)
245. print(f"AdaBoost Best Params: {adaboost_best}, Score: {adaboost_score}")
246.
247. base_model = RandomForestClassifier(max_depth=5, random_state=42)
248. adaboost_60_40 = AdaBoostClassifier(learning_rate=0.1,
   estimator=base_model, n_estimators=50, algorithm="SAMME")
249. adaboost_60_40.fit(X_train, y_train)
250.
251. y_pred_adaboost = adaboost_60_40.predict(X_test)
252. print("AdaBoost 60:40 Results:")
253. results['AdaBoost_60_40'] = evaluate_metrics(y_test, y_pred_adaboost)
```

The X_train and y_train variables are used to find the best parameters using catboost_best_params function at line 232, xgboost_best_params function at line 235, and adaboost_best_params function at line 244. Then after the best parameters are found, the parameters are used to train the model with X_train and y_train variables and to the X_test variable. After the result is out, we evaluate the classification for each model at line 233, 242, and 253 using the evaluate_metrics function by inputting the y_test which is the real value from the dataset and the classificate value from model classification.

Results

In this research, researchers tried several splitting sizes to three algorithms. Boosted Random Forest that consists of AdaBoost Classifier and Random Forest Classifier as the base model, XGBoost Classifier, and CatBoost Classifier. After researchers found the best parameters and then found accuracy, precision, recall, and F1 score, we compared the values of each splitting method. In this study, researchers take the highest accuracy from each splitting size.

Best Parameter Result

Tabel 2. Best Parameters

Models	Splitting 70:30	Splitting 80:20	Splitting 60:40
Boosted Random Forest (AdaBoost Classifier and Random Forest Classifier)	learning_rate: 0.1, max_depth: 5, n_estimators: 50	learning_rate: 0.5, max_depth: 5, n_estimators: 25	learning_rate: 0.5, max_depth: 6, n_estimators: 50
CatBoost Classifier	learning_rate: 1, depth: 10, iterations: 400	learning_rate: 0.1, depth: 10, iterations: 200	learning_rate: 0.5, depth: 5, iterations: 1000
XGBoost Classifier	learning_rate: 0.1, max_depth: 9, n_estimators: 400	learning_rate: 0.5, max_depth: 10, n_estimators: 500	learning_rate: 0.1, max_depth: 8, n_estimators: 100

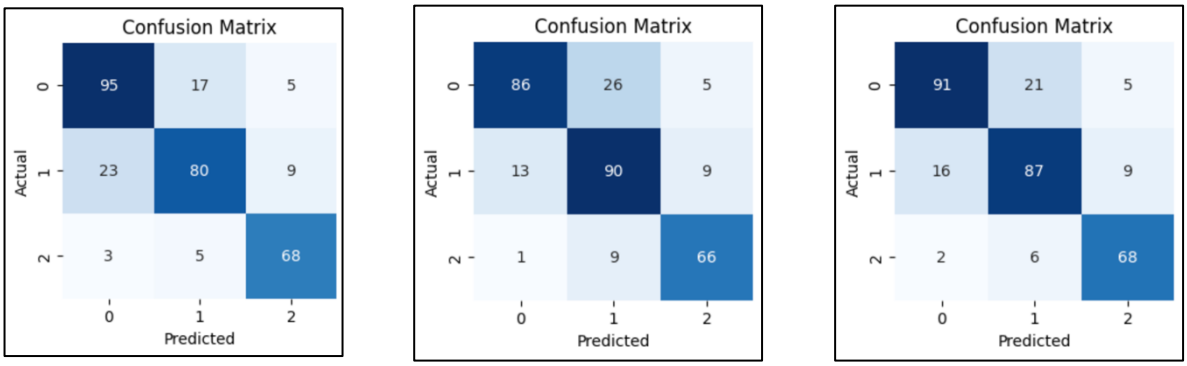
After training and cross validating many combinations of parameters with specific range of values, this research found the best parameters for each algorithm and each data splitting size that can be seen from Tabel 2. There are 9 best parameters that have been found.

Result For Splitting 70:30

Tabel 3. Splitting 70:30 Results

Parameters	Boosted Random Forest	CatBoost Classifier	XGBoost Classifier
Accuracy	0.797	0.793	0.806
Precision	0.8	0.802	0.809
Recall	0.806	0.802	0.816
F1 Score	0.802	0.799	0.812

From Tabel 3, we conclude that XGBoost Classifier has the highest accuracy of 80.7% and Boosted Random Forest and CatBoost Classifier has the lowest accuracy of 79.3%. It means that XGBoost Classifier can predict the class better than the rest of the models. It can be seen at the confusion matrix at Gambar 4, that CatBoost Classifier have the least amount of True Positive and XGBoost Classifier most amount of True Positives. CatBoost Classifier has 242 True Positives, Boosted Random Forest has 243 True Positives, and XGBoost Classifier has 246 True Positives.



(a) (b) (c)

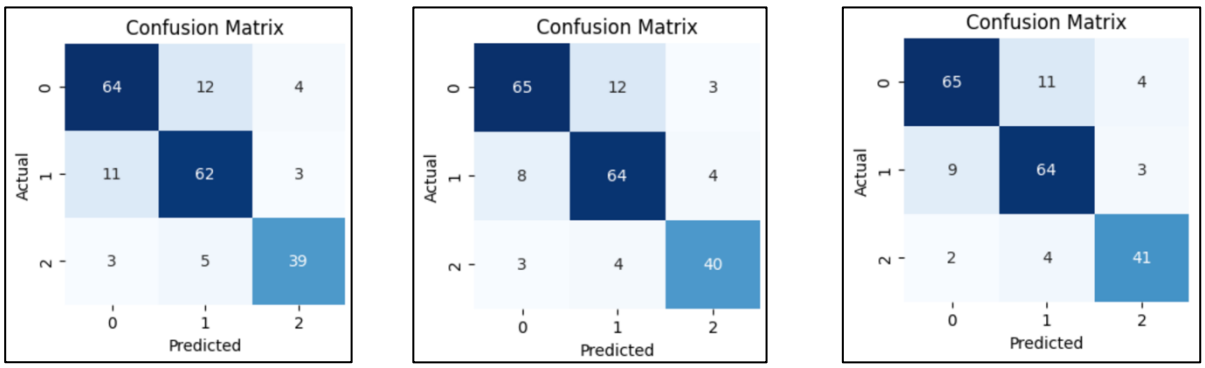
Gambar 4. (a) CM 70:30 Boosted Random Forest, (b) CM 70:30 CatBoost Classifier, (c) CM 70:30 XGBoost Classifier

Result For Splitting 80:20

Tabel 4. Splitting 80:20 Results

Parameters	Boosted Random Forest	CatBoost Classifier	XGBoost Classifier
Accuracy	0.813	0.813	0.813
Precision	0.832	0.832	0.832
Recall	0.837	0.837	0.837
F1 Score	0.818	0.818	0.818

From Tabel 4, we conclude that XGBoost Classifier has the highest accuracy of 83.7% and Boosted Random Forest has the lowest accuracy of 81.8%. It means that XGBoost Classifier can predict the class better than the rest of the models. It can be seen at the confusion matrix at Gambar 5, that Boosted Random Forest has the least amount of True Positive and XGBoost Classifier has the most amount of True Positives. Boosted Random Forest has 165 True Positive, CatBoost Classifier has 169, and XGBoost Classifier has 170 True Positive.



(a) (b) (c)

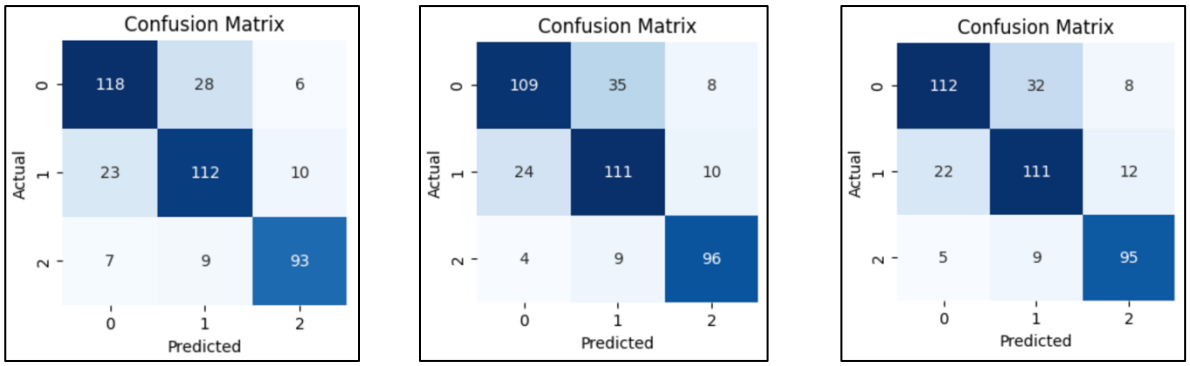
Gambar 5. (a) CM 80:20 Boosted Random Forest, (b) CM 80:20 CatBoost Classifier, (c) CM 80:20 XGBoost Classifier

Result For Splitting 60:40

Tabel 5. Splitting 60:40 Results

Parameters	Boosted Random Forest	CatBoost Classifier	XGBoost Classifier
Accuracy	0.796	0.796	0.796
Precision	0.778	0.778	0.778
Recall	0.783	0.783	0.783
F1 Score	0.8	0.8	0.8

From Tabel 5, we conclude that Boosted Random Forest has the highest accuracy of 79.6% and CatBoost Classifier has the lowest accuracy of 77.8%. It means that Boosted Random Forest can predict the class better than the rest of the models. It can be seen at the confusion matrix at Gambar 6, that CatBoost Classifier has the least amount of True Positive and Boosted Random Forest has the most amount of True Positives. CatBoost Classifier has 316 True Positive, XGBoost Classifier has 318, and Boosted Random Forest has 323 True Positive.



(a) (b) (c)

Gambar 6. (a) CM 60:40 Boosted Random Forest, (b) CM 60:40 CatBoost Classifier, (c) CM 60:40 XGBoost Classifier

Discussion

In this study, researchers used several sizes of data splitting and several algorithms. Based on the algorithm, The Boosted Random Forest has the highest accuracy of 81.3% by using the 80:20 splitting and the lowest accuracy of 79.6% by using the 60:40 splitting size. CatBoost Classifier has the highest accuracy of 83.2% by using the 80:20 splitting and the lowest accuracy of 77.8% by using the 60:40 splitting size. The XGBoost Classifier has the highest accuracy of 83.7% by using the 80:20 splitting and the lowest accuracy of 78.3% by using the 60:40 splitting size. Even though the Boosted Random Forest is more complex than the other algorithms because it used a Random Forest Classifier as the base model and the other algorithms used a Decision Tree Classifier, it still has a lower accuracy than XGBoost Classifier. It means that XGBoost is more suitable for this dataset.

Both XGBoost Classifier and CatBoost Classifier are based on gradient boosting algorithm. In this research, XGBoost achieves a higher classification accuracy than CatBoost. XGBoost is superior due to its efficient split finding algorithm for numerical features. While CatBoost's main advantage in handling category data is irrelevant because the dataset doesn't have a categorical feature. XGBoost uses L1 and L2 regularization to control model complexity, while CatBoost uses only simple regularization during tree formation. Regularization in XGBoost helps prevent overfitting while maintaining flexibility in modeling numerical data.

Based on the splitting, there are three sizes. The splits are 70% for data training and 30% for data testing, 80% for data training and 20% for data testing, and 60% for data training and 40% for data testing. The 70% and 30% splitting receives the accuracy of 79.7% , 79.3%, and 80.6%. So the rate of accuracy of the model by using the 70% and 30% splitting is 79.9%. Then 80% and 20% splitting receives the accuracy of 81.3% , 83.2%, and 83.7%. So the rate of accuracy of the model by using the 80% and 20% splitting is 82.7%. Then 60% and 40% splitting receives the accuracy of 79.6% , 77.8%, and 78.3%. So the rate of accuracy of the model by using the 60% and 40% splitting is 78.6%.

From the results, we can conclude several things. First, from all algorithms, the XGBoost Classifier that uses the data size of 80% for data training and 20% for data testing has the highest result of accuracy by 84%. Second, the 80:20 splitting size has the highest accuracy rate than the other size which is 82.73%. This means that the 80:20 splitting size is the most suitable size for those 3 algorithms with the specific or limited parameters. Thirdly, the 60:40 splitting size has the lowest accuracy rate than the other size which is 78.57%. It can happen because the data for training is too small for the model/algorithm to learn the data.

CONCLUSION

Based on the results that have been obtained from the research, it can be seen that the accuracy of each algorithm with different dataset splitting sizes is higher than 75%. This means that the boosting algorithms that have been used in this research can classify maternal health risks. After several experiments, the XGBoost Classifier model that uses the data splitting size of 80% for training and 20% for testing has a better performance than the rest of the models. It has the highest accuracy, precision, recall, and F1-scores than the rest of the models.

Based on the splitting, the 80% data for training and 20% for testing has the highest accuracy rate than the rest of the algorithms and the 60% data for training and 40% for testing has the lowest accuracy rate than the rest of the algorithms. This means that the data splitting of 80% for training and 20% for testing is more suitable for the model that has been used in this research.

For future studies, researchers can add or use a new dataset from many sources. A variety of ensemble methodologies that have not been explored are also available. There are also several health metrics that can be added into the dataset, like vaccination history, nutritional status, and others.

REFERENCES

[1] R. Musarandega, M. Nyakura, R. Machekano, R. Pattinson, and S. P. Munjanja, "Causes of maternal mortality in Sub-Saharan Africa: A systematic review of studies published from 2015 to 2020," *J Glob Health*, vol. 11, p. 04048, 2021, doi: 10.7189/jogh.11.04048

[2] O. Olonade, T. I. Olawande, O. J. Alabi, and D. Imhonopi, "Maternal Mortality and Maternal Health Care in Nigeria: Implications for Socio-Economic Development," *Open Access Maced J Med Sci*, vol. 7, no. 5, pp. 849–855, Mar. 2019, doi: 10.3889/oamjms.2019.041

[3] "Maternal mortality," *WHO*, Feb. 22, 2023. Available: https://www.who.int/news-room/fact-sheets/detail/maternal-mortality

[4] M. Y. Al-Hindi *et al.*, "Association of Antenatal Risk Score With Maternal and Neonatal Mortality and Morbidity," *Cureus*, vol. 12, no. 12, p. e12230, Dec. 2020, doi: 10.7759/cureus.12230

[5] T. O. Togunwa, A. O. Babatunde, and K.-R. Abdullah, "Deep hybrid model for maternal health risk classification in pregnancy: synergy of ANN and random forest," *Front. Artif. Intell.*, vol. 6, Jul. 2023, doi: 10.3389/frai.2023.1213436. Available: https://www.frontiersin.org/articles/10.3389/frai.2023.1213436. [Accessed: Apr. 02, 2024]

[6] I. D. Mienye and Y. Sun, "A Survey of Ensemble Learning: Concepts, Algorithms, Applications, and Prospects," *IEEE Access*, vol. 10, pp. 99129–99149, 2022, doi: 10.1109/ACCESS.2022.3207287

[7] K. Banujan, M. Ifham, and B. T. G. S. Kumara, "Boosting Ensemble Machine Learning Approach for Covid-19 Death Prediction," vol. 3, no. 1, Art. no. 1, Feb. 2023, doi: 10.4038/sljssh.v3i1.88

[8] C. Iwendi *et al.*, "COVID-19 Patient Health Prediction Using Boosted Random Forest Algorithm," *Front Public Health*, vol. 8, p. 357, Jul. 2020, doi: 10.3389/fpubh.2020.00357

[9] Y. Mishina, R. Murata, Y. Yamauchi, T. Yamashita, and H. Fujiyoshi, "Boosted Random Forest," *IEICE Transactions on Information and Systems*, vol. E98.D, no. 9, pp. 1630–1636, 2015, doi: 10.1587/transinf.2014OPP0004

[10] J. Sanjaya, E. Renata, V. E. Budiman, F. Anderson, and M. Ayub, "Prediksi Kelalaian Pinjaman Bank Menggunakan Random Forest dan Adaptive Boosting," *Jurnal Teknik Informatika dan Sistem Informasi*, vol. 6, no. 1, Art. no. 1, Apr. 2020, doi: 10.28932/jutisi.v6i1.2313. Available: https://journal.maranatha.edu/index.php/jutisi/article/view/2313. [Accessed: Apr. 04, 2024]

[11] I. Abuelezz *et al.*, "Contribution of Artificial Intelligence in Pregnancy: A Scoping Review," *Stud Health Technol Inform*, vol. 289, pp. 333–336, Jan. 2022, doi: 10.3233/SHTI210927

[12] A. Fauzi, E. Utami, and A. D. Hartanto, "Penerapan Random Forest dan Adaboost untuk Klasifikasi Serangan DDoS," *Journal on Education*, vol. 5, no. 3, Art. no. 3, Feb. 2023, doi: 10.31004/joe.v5i3.1582

[13] V. Srivardhan, "Adaptive boosting of random forest algorithm for automatic petrophysical interpretation of well logs," *Acta Geod Geophys*, vol. 57, no. 3, pp. 495–508, Sep. 2022, doi: 10.1007/s40328-022-00385-5

[14] M. Ahmed, M. A. Kashem, M. Rahman, and S. Khatun, "Review and Analysis of Risk Factor of Maternal Health in Remote Area Using the Internet of Things (IoT)," in *InECCE2019*, A. N. Kasruddin Nasir, M. A. Ahmad, M. S. Najib, Y. Abdul Wahab, N. A. Othman, N. M. Abd Ghani, A. Irawan, S. Khatun, R. M. T. Raja Ismail, M. M. Saari, M. R. Daud, and A. A. Mohd Faudzi, Eds., Singapore: Springer, 2020, pp. 357–365. doi: 10.1007/978-981-15-2317-5_30