Applied Medical Informatics
Original Research Vol. 45, No. 2/2023, pp: 70-81

Optimization of Machine Learning Algorithms with Bagging and AdaBoost Methods for Stroke Disease Prediction

Helmi Saifullah MANSUR, Nelly Oktavia ADIWIJAYA*, Tio DHARMAWAN

Department of Computer Science, University of Jember, Jalan Kalimantan No. 37 Jember, East Java, 68121, Indonesia.
E-mail: helmimansur88@gmail.com; (*) nelly.oa@unej.ac.id; tio.pssi@unej.ac.id

* Author to whom correspondence should be addressed;

Received: May 25, 2023 /Accepted: June 25, 2023/ Published online: July 1, 2023

Abstract
Stroke is an acute neurologic disorder of blood vessels in the brain due to blockage of blood flow to the brain resulting in less oxygen. Stroke remains one of the leading causes of death worldwide. Therefore, developing Machine Learning is expected to help health professionals make early predictions of stroke. This study aimed to compare the performance results of stroke classification modeling using Bagging and AdaBoost methods in Machine Learning algorithms (Naïve Bayes, Support Vector Machine, Decision Tree, and K-Nearest Neighbors) using Stroke Prediction Dataset from Kaggle. The results show that Machine Learning algorithm that has the best performance is Decision Tree with 91% accuracy, followed by KNN, Naïve Bayes, and finally, SVM. Optimization of Machine Learning algorithms with Bagging and AdaBoost only increases the performance value of the Decision Tree algorithm but does not increase the performance value of other algorithms. The results of Decision Tree optimization with Bagging increased 1% accuracy and F1-score, as well as 4% precision in the missing value deleted scenario. Furthermore, in the missing value scenario using mean value increases 1% F1-score and 4% precision. While the results of Decision Tree optimization with AdaBoost increased 1% recall and F1-score in the missing value deleted scenario. Then in the missing value scenario using mean value has the same performance as without optimization. The application of Bagging and AdaBoost methods only increases the performance value of the Decision Tree algorithm, but the increase is still insignificant.

Keywords: Stroke; Machine Learning (ML); Bagging; AdaBoost

Introduction

Stroke is an acute neurological dysfunction of the blood vessels in the brain caused by the cessation of blood supply to the brain so that brain cells lack the necessary oxygen [1]. Based on the 2019 Global Burden of Disease (GBD) information, stroke remains the second leading cause of death and the third leading cause of combined death and disability in the world [2]. Basic Health Research data in Indonesia stated that in 2013 was 12.1 per mile of national stroke prevalence, while in 2018, it reported a prevalence of 10.9 per mile population, with the highest values in East Kalimantan Province (14.7 per mile) and the lowest in Papua Province (4.1 per mile) [3]. Based on its type, stroke can be ischemic stroke or hemorrhagic stroke [4]. Ischemic stroke occurs due to blockage of blood vessels by a thrombus or embolus, resulting in brain ischemia, while hemorrhagic stroke occurs due to bleeding and rupture of weakened blood vessels around the brain tissue, causing intracranial pressure [5].

With the development of information and communication technology in both the fields of Artificial Intelligence (AI) and Machine Learning (ML), it is hoped that it can take an important role in making early predictions to treat various diseases, one of which is stroke [1]. Machine learning techniques have been widely used in multiple healthcare applications in recent years [6]. Machine learning can be a useful step towards efficient treatment in early stroke detection and assist healthcare professionals in making clinical decisions and predictions. Research in the last few decades, machine learning has been used in improving stroke diagnosis in terms of accuracy and speed [5].

Sailasya and Kumari [7] compared six machine learning algorithms (Logistic Regression, Decision Tree, Random Forest, K Nearest Neighbor, Support Vector Machine, and Naïve Bayes) and data balancing with undersampling techniques on the stroke prediction dataset from Kaggle [8], showing that the Naïve Bayes algorithm has the best performance with an accuracy value of 82%. Dritsas and Trigka, reported on the Kaggle stroke dataset [8], that the results using SMOTE and the Stacking Classification method have good results with 98% accuracy [1].

Byna and Basit conducted qualitative research using questionnaires and analyzed secondary data from Banjarmasin Regional Hospital and showed that AdaBoost on the Naïve Bayes algorithm provides optimization of 0.005 with an accuracy of 98.1% [9]. Another study conducted on the Kaggle stroke dataset, without class balancing on the data, reported accuracy of classification equal to 92.87% K-fold (K-10) cross-validation, 95.02% for the C4.5 + Bagging algorithm, and 94.63% accuracy for the C4.5 + AdaBoost algorithm [10].

Combining different classification methods can perform better than a single prediction model and produces accurate predictions [11]. No comparison of the Bagging and AdaBoost ensemble methods using Machine Learning algorithms has been reported on the Kaggle stroke dataset [8]. Our research objective was to compare the performance results of stroke classification modeling using Bagging and AdaBoost methods so as to improve the performance and efficiency of Machine Learning algorithms in solving certain problems. The results of this study are expected to help health experts make decisions in predicting early stroke disease.

Material and Method

Our research was conducted at the Faculty of Computer Science, University of Jember using Google Collaboratory on an Acer Aspire E5 421 laptop with AMD A6-6310 Processor. Figure 1 shows the research stages that became the reference for researchers in conducting research.

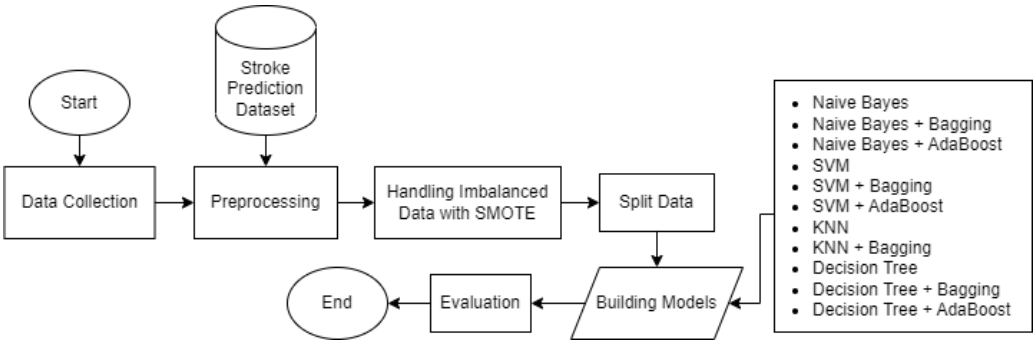

Figure 1. Research Stages

Data Collection

The dataset used is the Kaggle stroke dataset [8]. The database is freely available as *.csv file and was downloaded on 10 January 2023. The dataset has 5110 total data with 12 variables, including 11 independent variables and one dependent variable (Table 1). The dependent variable of this dataset is the variable 'stroke' with 245 samples identified as stroke and 4861 samples identified as not stroke.

Table 2 shows details related to the raw data in the top 5 rows of the dataset before preprocessing.

Table 1. Variable description in the Kaggle stroke dataset

Variable	Data Type	Description
id	int64	Unique number of each patient
gender	object (Male, Female, Other)	Patient gender
age	float64	Patient age
hypertension	int64 (1, 0)	Presence (1) or absence (0) of hypertension
heart_disease	int64 (1, 0)	Presence (1) or absence (0) of heart disease
ever_married	object (Yes, No)	Patient marital status: ever married (Yes) vs. never married (No)
work_type	object (children, Govt_job, Never_worked, Private, Self-employed)	Job category of each patient
residence_type	object (Urban, Rural)	Type of residence
avg_glucose_level	float64	Average blood glucose level of each patient
bmi	float64	Body mass index
smoking_status	object (formerly smoked, never smoked, smokes, unknown)	Smoking status
stroke	int64 (1, 0)	Output column that provides the status of whether the patient is identified (1) or not (0) with stroke

Table 2. Raw data details of the top 5 rows of the dataset

id	gender	age	hypertension	heart_disease	ever_married
9046	Male	67.0	0	1	Yes
51676	Female	61.0	0	0	Yes
31112	Male	80.0	0	1	Yes
60182	Female	49.0	0	0	Yes
1665	Female	79.0	1	0	Yes
work_type	**residence_type**	**avg_glucose_level**	**bmi**	**smoking_status**	**stroke**
Private	Urban	228.69	36.6	formerly smoked	1
Self-employed	Rural	202.21	NaN	never smoked	1
Private	Rural	105.92	32.5	never smoked	1
Private	Urban	171.23	34.4	smokes	1
Self-employed	Rural	174.12	24.0	never smoked	1

Preprocessing

The preprocessing steps were carried out to clean and normalize the data. Table 3 shows the scenario of the preprocessing steps in the research.

Table 3. The scenario of preprocessing step

Steps	Treatment	Description
Handling Missing Values	1. Removing missing values in the data with dropna() 2. Fill in missing values with the mean according to research [7,9]	Missing values in the dataset are found in the 'bmi' column, with 201 missing values (3.93%).
Removing Unnecessary Columns	Remove the 'id' column with the drop function	The 'id' column is removed as it has no impact and relationship in model building.

Steps	Treatment	Description
Remove the 'Other' value in the 'Gender' column	Remove the 'Other' value with the drop function	The value 'Other' was deleted because there was only 1 data.
Label Encoding Data	Label encoding using the LabelEncoder() function	The five columns that need label encoding include 'gender', 'ever_married', 'work_type', 'residence_type', and 'smoking_status'.

In this stage, we did the following steps:

a. Handling Missing Values
Two-hundred and one values were missing in the 'bmi' (3.93%). Missing values in the 'bmi' column are handled with 2 scenarios, which are:
   1. Deleted by using the dropna() function. This needs to be done because missing values can cause bias in data analysis and inaccurate results, so removing missing values will make the results reliable and accurate. The amount of data after missing values are removed encounted 4909 data.
   2. Using the mean value with the fillna() function. The reason for using the mean value is because the mean describes the average value of a set of data so that it can provide general data information. Using the mean value in handling missing values also helps in maintaining the shape of the data distribution because the missing data is assumed to have a similar distribution to the available data.

b. Removing Unnecessary Columns
In this research, the 'id' column is removed because the 'id' column only contains the unique number of each patient and has no impact on model building.

b. Remove the 'Other' value in the 'Gender' column
The 'Other' value in the 'gender' column was removed. Only 1 data had this value, so the case is contribute less. The amount of data after the 'Other' value is removed for the missing value scenario using the mean value is 5109 data and for the missing value scenario deleted is 4908 data.

c. Label Encoding Data
At this step, categorical columns are converted into numeric columns so that the model building process can be carried out using the LabelEncoder() function from the sklearn.preprocessing library. Table 4 shows the details of data value changes in each categorical column.

Table 4. Details of Data Value Changes in Each Categorical Column

Categorical Columns	Initial Data Value	Conversion Data Value
gender	Female	0
	Male	1
ever_married	Yes	0
	No	1
work_type	Govt_job	0
	Never_worked	1
	Private	2
	Self-employed	3
	children	4
residence_type	Rural	0
	Urban	1
smoking_status	Unknown	0
	formerly smoked	1
	never smoked	2
	smokes	3

Handling Imbalanced Data with Synthetic Minority Oversampling Technique

The target class column in the dataset, 'stroke', shows data imbalance with 209 rows indicating the occurrence of stroke and 4699 rows having the possibility of no stroke for the missing value

deleted scenario. In the missing value scenario using the mean value also shows imbalance in the target class data with 249 rows indicating stroke and 4860 rows having the possibility of no stroke. Therefore, it is necessary to handle imbalanced data so that the model provides accurate results and efficient predictions. In this research, the handling is done with the SMOTE (Synthetic Minority Oversampling Technique) technique using the SMOTE() function from the imblearn.over_sampling library. SMOTE applies An oversampling approach is applied to minority classes by synthesizing new minority class samples from several adjacent minority classes [12].

Split Data

In this research, we splitting the available data into training and testing sets using k-fold cross validation with a k-fold value of 10. K-fold cross validation is a method used to evaluate model performance by dividing data into k equal subsets with a specified k value.

10-Fold Cross Validation means that data that has been divided into ten equal subsets will be iterated to evaluate the model with nine subsets (90% of the data) used as training data and the remaining one subset (10% of the data) used as testing data. This is repeated until all subsets become testing data. The final model used in K-Fold Cross Validation is the model that has the best evaluation score performance [22].

Building Models

Models were built with four machine-learning classification algorithms including Naïve Bayes, Support Vector Machine, Decision Tree, and K-Nearest Neigbors. These algorithms were chosen for comparison because they are popular types of binary classification algorithms that are widely used and have their own characteristics and advantages in classifying data.

Each algorithm is optimized with Bagging and AdaBoost techniques to determine which classification gives the best results so as to improve the performance and efficiency of Machine Learning algorithms in solving problems. The model is built using the functions provided from the sklearn library with the python language in Google Colaboratory. Table 5 shows the scenario of the model-building stage.

Table 5. Function scenarios used in the model building step

Model	Function in sklearn library	Description
Naïve Bayes	GaussianNB()	From the library sklearn.naive_bayes
Support Vector Machine (SVM)	SVC(probability=True)	From the library sklearn.svm
K Nearest Neigbor (KNN)	KNeighborsClassifier()	From the library sklearn.neighbors
Decision Tree	DecisionTreeClassifier()	From the library sklearn.tree
Naïve Bayes + Bagging	BaggingClassifier(base_estimator = GaussianNB(), max_sample=0.5, max_features=0.5)	max_sample is the maximum number of samples used to build the model in the bagging technique and max_features is the number of features to be used in each sub-model in the bagging technique.
Support Vector Machine (SVM) + Bagging	BaggingClassifier(base_estimator = SVC(probability=True), max_sample=0.5, max_features=0.5)	
K Nearest Neigbor (KNN) + Bagging	BaggingClassifier(base_estimator = KNeighborsClassifier(), max_sample=0.5, max_features=0.5)	
Decision Tree+ Bagging	BaggingClassifier(base_estimator = DecisionTreeClassifier(), max_sample=0.5, max_features=0.5)	
Naïve Bayes + AdaBoost	AdaBoostClassifier(base_estimator = GaussianNB(), n_ estimator=50)	KNN is not optimized with AdaBoost because KNN does not have sample weights in sklearn. n_estimator is used to
Support Vector Machine (SVM) + AdaBoost	AdaBoostClassifier(base_estimator = SVC(probability=True), n_estimator=50)	

Decision Tree+ AdaBoost	AdaBoostClassifier(base_estimator = DecisionTreeClassifier(), n_estimator=50)	control the number of iterations to be performed in the AdaBoost process.

Each model presented in Table 5 was applied to each dataset, so the calculations were done 10 times.

The Naïve Bayes algorithm assumes that the variables or attributes are independent or have no dependence on each other [13].

In the classification process, SVM builds a model using several support vectors or data points closest to the hyperplane to determine the dividing line [14].

K Nearest Neighbor (KNN) is a classification method that determines objects based on the distance of the learning data that is closest to the object [15,16].

Decision Tree is a classification method that applies decision rules by partitioning large data into smaller data sets [17,18].

Bagging is built using two main techniques: bootstrapping and aggregation by manipulating or duplicating the training data to build bagged classifiers [19,20].

AdaBoost focuses on adding models sequentially and then the next model corrects the predictions made by the previous model in the sequence [21].

Evaluation

Confusion matrix, accuracy, precision, recall, and F1-score were used to evaluate the algorithms. Confusion matrix is a matrix used in evaluating the performance of a classification model by describing the relationship between the predicted value and the actual value of a set of data. Table 6 is the calculation of the distribution of values on the confusion matrix.

Table 6. Confusion matrix

		Predicted Condition	
		Negative	**Positive**
Actual Condition	**Negative**	True Negative (TN)	False Positive (FP)
	Positive	False Negative (FN)	True Positive (TP)

Accuracy is used to measure the accuracy of the algorithm in predicting the right category in the classification or the percentage value of correct predictions from the total predictions made. The accuracy value is obtained from Eq. 1:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{FP} + \text{TN}} \quad (1)$$

Precision is used to measure how reliable the classification model is or the value of the number of correct predictions out of the total number of correct and incorrect predictions made by the model. The precision value is obtained from Eq. 2:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (2)$$

Recall is used to measure how well the classification model captures all correct predictions or the value of the number of correct predictions made by the model divided by the number of correct predictions that the model should make. The recall value is obtained from Eq. 3:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (3)$$

F1-score describes the model's overall quality using the combined value of precision and recall. The F1-score value is obtained from Eq. 4:

$$\text{F1 score} = \frac{2 * (\text{Recall} * \text{Precision})}{\text{Recall} + \text{Precision}} \quad (4)$$

Results

Handling Imbalanced Data with Synthetic Minority Oversampling Technique

Figure 2 shows a diagram of the comparison of the amount of data in the target class.

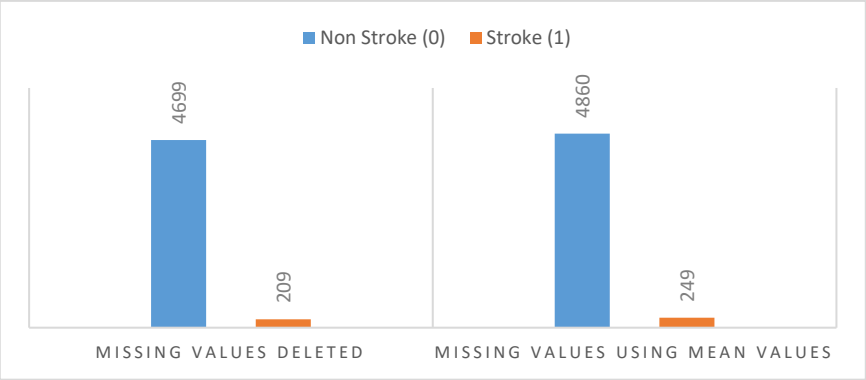

Figure 2. Diagram of the number of data in the target class

The results after balancing the data in the target class, the values '0' and '1' have the same amount of data and are balanced. Figure 3 shows a diagram of the comparison of the amount of data in the target class after being handled with SMOTE.

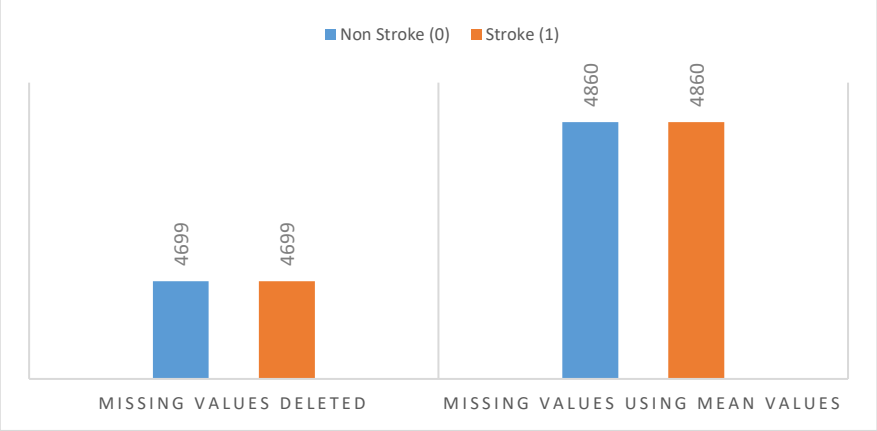

Figure 3. Diagram of the number of data in the target class after handling with smote

Models Evaluation

The results show that the performance of the algorithm has almost the same results both with the missing value deleted scenario and the missing value using the mean value. Table 7 and Table 8 show the confusion matrix for each scenario.

Table 7. Confusion matrix scenario missing value deleted

			Predicted Condition			
			With SMOTE		Without SMOTE	
			Non Stroke	Stroke	Non Stroke	Stroke
A	Naïve Bayes	Non Stroke	3312	1387	4218	481

			Predicted Condition			
			With SMOTE		Without SMOTE	
			Non Stroke	Stroke	Non Stroke	Stroke
Actual Condition		Stroke	940	3759	209	0
	Naïve Bayes + Bagging	Non Stroke	3264	1435	4458	241
		Stroke	1110	3589	209	0
	Naïve Bayes + AdaBoost	Non Stroke	2054	2645	3937	762
		Stroke	1421	3287	209	0
	SVM	Non Stroke	3095	1604	4699	0
		Stroke	1086	3613	209	0
	SVM + Bagging	Non Stroke	3032	1667	4699	0
		Stroke	1062	3637	209	0
	SVM + AdaBoost	Non Stroke	2960	1739	4699	0
		Stroke	1513	3186	209	0
	KNN	Non Stroke	3730	969	4660	39
		Stroke	90	4609	209	0
	KNN + Bagging	Non Stroke	3658	1041	4699	0
		Stroke	338	4361	209	0
	Decision Tree	Non Stroke	4210	489	4480	219
		Stroke	353	4346	209	0
	Decision Tree + Bagging	Non Stroke	4409	290	4698	1
		Stroke	415	4284	209	0
	Decision Tree + AdaBoost	Non Stroke	4227	472	4470	229
		Stroke	345	4354	209	0

Table 8. Confusion matrix when the mean value was used for of missing values

			Predicted Condition			
			With SMOTE		Without SMOTE	
			Non Stroke	Stroke	Non Stroke	Stroke
Actual Condition	Naïve Bayes	Non Stroke	3424	1436	4350	510
		Stroke	812	4048	249	0
	Naïve Bayes + Bagging	Non Stroke	3338	1522	4620	240
		Stroke	960	3900	249	0
	Naïve Bayes + AdaBoost	Non Stroke	3256	1604	3383	1477
		Stroke	2343	2517	249	0
	SVM	Non Stroke	3224	1636	4860	0
		Stroke	1033	3827	249	0
	SVM + Bagging	Non Stroke	3168	1692	4860	0
		Stroke	1021	3839	249	0
	SVM + AdaBoost	Non Stroke	3621	1239	4860	0
		Stroke	2291	2569	249	0
	KNN	Non Stroke	3846	1014	4803	57
		Stroke	106	4754	249	0
	KNN + Bagging	Non Stroke	3791	1069	4860	0
		Stroke	350	4510	249	0
	Decision Tree	Non Stroke	4325	535	4621	239
		Stroke	360	4500	249	0
	Decision Tree + Bagging	Non Stroke	4512	348	4854	6
		Stroke	512	4348	249	0
	Decision Tree + AdaBoost	Non Stroke	4324	536	4610	250
		Stroke	371	4489	249	0

Results of the distribution of actual condition values and predicted conditions in the confusion matrix Table 7 and Table 8 have almost the same distribution value for each scenario. From these

results, it is known that the confusion matrix model that does not handle imbalanced data with SMOTE has a biased distribution which is indicated by a model that tends to predict the majority class and misclassify the minority class. This happens because there is an unbalanced distribution in the target class so that the model is less adequate in recognizing patterns in the minority class. From the confusion matrix value, the model's accuracy, precision, recall, and F1-score can then be calculated. Table 9 and Table 10 below show the model performance results for each scenario.

Table 9. Model performance results scenario missing value deleted

Algorithm	With SMOTE				Without SMOTE			
	Accuracy %	Precision %	Recall%	F1-score%	Accuracy %	Precision %	Recall%	F1-score%
Naïve Bayes	75 (74.1, 75.8)	73 (72.1, 73.8)	80 (79.1, 80.8)	76 (75.1, 76.8)	86 (85.3, 86.6)	0 (0, 0)	0 (0, 0)	0 (0, 0)
Naïve Bayes + Bagging	73 (72.1, 73.8)	71 (70.0, 71.9)	76 (75.1, 76.8)	74 (73.1, 74.8)	91 (90.4, 91.5)	0 (0, 0)	0 (0, 0)	0 (0, 0)
Naïve Bayes + AdaBoost	57 (55.9, 58.0)	55 (53.9, 56.0)	70 (69.0, 70.9)	62 (61.0, 62.9)	80 (79.2, 80.7)	0 (0, 0)	0 (0, 0)	0 (0, 0)
SVM	71 (70.0, 71.9)	69 (68.0, 69.9)	77 (76.1, 77.8)	73 (72.1, 73.8)	96 (95.6, 96.3)	0 (0, 0)	0 (0, 0)	0 (0, 0)
SVM + Bagging	71 (70.0, 71.9)	69 (68.0, 69.9)	77 (76.1, 77.8)	73 (72.1, 73.8)	96 (95.6, 96.3)	0 (0, 0)	0 (0, 0)	0 (0, 0)
SVM + AdaBoost	65 (64.0, 65.9)	65 (64.0, 65.9)	68 (67.0, 68.9)	67 (66.0, 67.9)	96 (95.6, 96.3)	0 (0, 0)	0 (0, 0)	0 (0, 0)
KNN	89 (88.3, 89.6)	83 (82.2, 83.7)	98 (97.7, 98.2)	90 (89.3, 90.6)	95 (94.5, 95.4)	0 (0, 0)	0 (0, 0)	0 (0, 0)
KNN + Bagging	85 (84.2, 85.7)	81 (80.2, 81.7)	93 (92.4, 93.5)	87 (86.3, 87.6)	96 (95.6, 96.3)	0 (0, 0)	0 (0, 0)	0 (0, 0)
Decision Tree	91 (90.4, 91.5)	90 (89.3, 90.6)	92 (91.4, 92.5)	91 (90.4, 91.5)	91 (90.4, 91.5)	0 (0, 0)	0 (0, 0)	0 (0, 0)
Decision Tree + Bagging	**92 (91.4, 92.5)**	**94 (93.5, 94.4)**	**91 (90.4, 91.5)**	**92 (91.4, 92.5)**	96 (95.6, 96.3)	0 (0, 0)	0 (0, 0)	0 (0, 0)
Decision Tree + AdaBoost	91 (90.4, 91.5)	90 (89.3, 90.6)	93 (92.4, 93.5)	92 (91.4, 92.5)	91 (90.4, 91.5)	0 (0, 0)	0 (0, 0)	0 (0, 0)

Table 10. Model performance results of missing value scenario using mean value

Algorithm	With SMOTE				Without SMOTE			
	Accuracy %	Precision %	Recall%	F1-score%	Accuracy %	Precision %	Recall%	F1-score%
Naïve Bayes	77 (76.1, 77.8)	74 (73.1, 74.8)	83 (82.2, 83.7)	75 (74.1, 75.8)	85 (84.2, 85.7)	0 (0, 0)	0 (0, 0)	0 (0, 0)
Naïve Bayes + Bagging	75 (74.1, 75.8)	72 (71.1, 72.8)	80 (79.2, 80.7)	76 (75.1, 76.8)	90 (89.4, 90.5)	0 (0, 0)	0 (0, 0)	0 (0, 0)
Naïve Bayes + AdaBoost	59 (58.0, 59.9)	61 (60.0, 61.9)	52 (51.0, 52.9)	56 (55.0, 56.9)	66 (65.0, 66.9)	0 (0, 0)	0 (0, 0)	0 (0, 0)
SVM	73 (72.1, 73.8)	70 (69.0, 70.9)	79 (78.1, 79.8)	74 (73.1, 74.8)	95 (94.5, 95.4)	0 (0, 0)	0 (0, 0)	0 (0, 0)
SVM + Bagging	72 (71.1, 72.8)	69 (68.0, 69.9)	79 (78.1, 79.8)	74 (73.1, 74.8)	95 (94.5, 95.4)	0 (0, 0)	0 (0, 0)	0 (0, 0)
SVM + AdaBoost	64 (63.0, 64.9)	67 (66.0, 67.9)	53 (52.0, 53.9)	59 (58.0, 59.9)	95 (94.5, 95.4)	0 (0, 0)	0 (0, 0)	0 (0, 0)
KNN	88 (87.3, 88.6)	82 (81.2, 82.7)	98 (97.7, 98.2)	89 (88.3, 89.6)	94 (93.5, 94.4)	0 (0, 0)	0 (0, 0)	0 (0, 0)
KNN + Bagging	85 (84.2, 85.7)	81 (80.2, 81.7)	93 (92.4, 93.5)	87 (86.3, 87.6)	95 (94.5, 95.4)	0 (0, 0)	0 (0, 0)	0 (0, 0)

Algorithm	With SMOTE				Without SMOTE			
	Accuracy %	Precision %	Recall%	F1-score%	Accuracy %	Precision %	Recall%	F1-score%
Decision Tree	91 (90.4, 91.5)	89 (88.3, 89.6)	93 (92.4, 93.5)	91 (90.4, 91.5)	90 (89.4, 90.5)	0 (0, 0)	0 (0, 0)	0 (0, 0)
Decision Tree + Bagging	**91 (90.4, 91.5)**	**93 (92.4, 93.5)**	**90 (89.4, 90.5)**	**92 (91.4, 92.5)**	95 (94.5, 95.4)	0 (0, 0)	0 (0, 0)	0 (0, 0)
Decision Tree + AdaBoost	91 (90.4, 91.5)	89 (88.3, 89.6)	92 (91.4, 92.5)	91 (90.4, 91.5)	90 (89.4, 90.5)	0 (0, 0)	0 (0, 0)	0 (0, 0)

Discussion

The model performance results for each scenario (Table 9 and 10) show that Decision Tree algorithm has the best performance with 91% accuracy, followed by KNN algorithm with 89% accuracy for the missing value deleted scenario and 88% accuracy for the missing value scenario using mean value, Naïve Bayes algorithm with 75% accuracy for missing value deleted scenario and 77% accuracy for missing value scenario using mean value, then finally SVM algorithm with 71% accuracy for missing value deleted scenario and 73% accuracy for missing value scenario using mean value.

Our results differ from those reported by Sailasya and Kumari [7], who handle the imbalanced data using undersampling, with an accuracy of Naïve Bayes equal to 82% accuracy, while the lowest performance was in the Decision Tree algorithm with 66% accuracy. The performance of our results could be explained by the use of SMOTE in the imbalanced data, approach that balances data by synthesizing new samples in minority classes so as not to eliminate important information and be able to provide stronger decision results.

The results of implementing Machine Learning algorithm optimization with bagging and AdaBoost were found to only work on the Decision Tree algorithm. Meanwhile, for KNN, Naïve Bayes, and SVM algorithms cannot improve algorithm performance results. The results in Table 9 and 10 show that optimizing Decision Tree algorithm with Bagging can increase 1% accuracy and F1-Score, and 4% precision for the missing value deleted scenario. Then for the missing value scenario using the mean value can increase 1% F1-Score and 4% precision. Optimizing the Decision Tree algorithm with AdaBoost can increase 1% recall and F1-Score for the missing value deleted scenario. The same results as the performance without optimization were observed when the mean was used as input data in cases of missing values.

Saputri et al. [10] showed that the combination of the C4.5 algorithm with the bagging and AdaBoost methods could improve classification performance, but with lower values for sensitivity and F1-score. The lower performances could be explained by the absence of imbalanced treatment.

Our results showed that the application of the bagging method for Machine Learning algorithm optimization proved to be superior to the AdaBoost method. One of the factors that influence this is because in bagging the models built are trained independently and in parallel so that they tend to be more resistant to overfitting and tolerant of noise in the data. The bagging and AdaBoost methods do have their own advantages. Therefore, this research tests these two methods to find out the results when used for optimization by looking at the characteristics of the machine learning task at hand.

The factors that affect the optimization results of the Machine Learning algorithm with the Bagging and AdaBoost methods are as follows:

a. Dataset characteristics and quality
   Dataset characteristics and quality such as imbalanced data can affect optimization results and model performance. In this study, it is found that the optimization results and performance of models that are handled with SMOTE have unbiased evaluation and performance compared to models that are not handled with imbalanced data.
b. Suitability of the algorithm used
   Each algorithm has its own characteristics and limitations in solving classification problems. In this research, it was found that the application of the Bagging method for Machine Learning

algorithm optimization proved superior to the AdaBoost method.

This research has limitations such as not doing feature selection on the dataset because all existing features are included in the factors that affect a person affected by stroke. But it is hoped that in the next research can try to do feature selection on the dataset in order to find out the features or attributes that have the most influence on stroke so that they can provide better results on classification. Another limitation is that this research only uses one model to be used as a base estimator in the bagging and AdaBoost methods, so it is hoped that the next research can try the bbagging and AdaBoost methods with a combined base estimator from several models in order to further optimize the classification performance results. In addition, it has been found that the Decision Tree + Bagging model has the best performance in this study. Future research is expected to deploy the model so that it can be applied to predict new data so that it can help health professionals in predicting early stroke disease.

Conclusion

Decision Tree algorithm performs best, followed by KNN, Naïve Bayes, and SVM algorithms for stroke classification modeling. Handling imbalanced data with SMOTE also makes the precision, recall, and F1-score values better than without handling. The results of optimizing Machine Learning algorithms with bagging and AdaBoost were found in the used dataset to only work to improve the performance of the Decision Tree algorithm. Meanwhile, the KNN, Naïve Bayes, and SVM algorithms cannot improve the algorithm performance results.

Optimization of Decision Tree algorithm with bagging can increase 1% accuracy and F1-Score, and 4% precision for the missing value deleted scenario. Then for the missing value scenario using the mean value can increase 1% F1-Score and 4% precision. The optimization of Decision Tree algorithm with AdaBoost can increase 1% recall and F1-Score for the missing value deleted scenario. Then for the missing value scenario using the mean value has the same results as the performance without optimization.

Conflict of Interest

The authors declare that they have no conflict of interest.

References

1. Dritsas E, Trigka M. Stroke Risk Prediction with Machine Learning Techniques. Sensors. 2022; 22(13):4670. doi:10.3390/s22134670
2. Feigin VL, Brainin M, Norrving B, Martins S, Sacco RL, Hacke W, et al. World Stroke Organization (WSO): Global Stroke Fact Sheet 2022. Int J Stroke. 2022;17(1):18–29. doi:10.1007/s10554-020-02099-7
3. Kementerian Kesehatan Republik Indonesia. Laporan Riskesdas 2018 Nasional [Internet]. 2019 [cited 2023 Jan 17]. p. 674. Available from: https://kesmas.kemkes.go.id/assets/upload/dir_519d41d8cd98f00/files/Hasil-riskesdas-2018_1274.pdf
4. Kokkotis C, Giarmatzis G, Giannakou E, Moustakidis S, Tsatalas T, Tsiptsios D, et al. An Explainable Machine Learning Pipeline for Stroke Prediction on Imbalanced Data. Diagnostics. 2022; 12(10):2392. doi:10.3390/diagnostics12102392
5. Sirsat MS, Fermé E, Câmara J. Machine Learning for Brain Stroke: A Review. J Stroke Cerebrovasc Dis. 2020;29(10):105162. doi: 10.1016/j.jstrokecerebrovasdis.2020.105162
6. Chun M, Clarke R, Cairns BJ, Clifton D, Bennett D, Chen Y, et al. Stroke risk prediction using machine learning: A prospective cohort study of 0.5 million Chinese adults. J Am Med Informatics Assoc. 2021;28(8):1719–1727.
7. Sailasya G, Kumari GLA. Analyzing the Performance of Stroke Prediction using ML Classification Algorithms. Int J Adv Comput Sci Appl. 2021;12(6):539–545.

8. Fedesoriano. Stroke Prediction Dataset [Internet]. Kaggle; 2020. Available from: https://www.kaggle.com/fedesoriano/stroke-prediction-dataset
9. Byna A, Basit M. Penerapan Metode Adaboost Untuk Mengoptimasi Prediksi Penyakit Stroke Dengan Algoritma Naïve Bayes. J Sisfokom (Sistem Inf dan Komputer). 2020;9(3):407–411.
10. Saputri ND, Khalid K, Rolliawati D. Komparasi Penerapan Metode Bagging dan Adaboost pada Algoritma C4 . 5 untuk Prediksi Penyakit Stroke. 2022;11(September):567–577.
11. Jothi Prakash V, Karthikeyan NK. Enhanced Evolutionary Feature Selection and Ensemble Method for Cardiovascular Disease Prediction. Interdiscip Sci – Comput Life Sci [Internet]. 2021;13(3):389–412. doi:10.1007/s12539-021-00430-x
12. Saifudin A. Level Data Dan Algoritma Untuk Penanganan Ketidakseimbangan Kelas [Internet]. Pascal Books; 2022. Available from: https://books.google.co.id/books?id=MG6dEAAAQBAJ
13. Arhami M, Nasir M. Data Mining - Algoritma dan Implementasi [Internet]. 1st ed. Penerbit Andi; 2020. 226 p. Available from: https://books.google.co.id/books?id=AtcCEAAAQBAJ
14. Sianturi FA, Hasugian PM, Simangunsong A, Nadeak B, Sihotang HT. DATA MINING: Teori dan Aplikasi Weka [Internet]. IOCS Publisher; 2019. 200 p. (Edisi). Available from: https://books.google.co.id/books?id=MWcHEAAAQBAJ
15. Id ID. Machine Learning : Teori, Studi Kasus dan Implementasi Menggunakan Python [Internet]. Unri Press; 2021. 210 p. Available from: https://books.google.co.id/books?id=JvBPEAAAQBAJ
16. Hidayah U, Sifaunajah A. Cara Mudah Memahami Algoritma K-Nearest Neighbor Studi Kasus Visual Basic 6.0 [Internet]. Lembaga Penelitian dan Pengabdian kepada Masyarakat Universitas KH. A. Wahab Hasbullah; 2019. Available from: https://books.google.co.id/books?id=B6t-EAAAQBAJ
17. Nofriansyah D. Konsep Data Mining Vs Sistem Pendukung Keputusan [Internet]. Yogyakarta: Deepublish; 2015. 120 p. Available from: https://books.google.co.id/books?id=PoJyCAAAQBAJ
18. Kusrini, Luthfi ET. Algoritma Data Mining [Internet]. 1st ed. Prabawati TA, editor. Yogyakarta: Penerbit Andi; 2009. 216 p. Available from: https://books.google.co.id/books?id=-Ojclag73O8C
19. Fitriyani. Metode Bagging Untuk Imbalance Class Pada Bedah Toraks Menggunakan Naïve Bayes. 2018;18(3):270–282.
20. Ridwan A, Khoiriyah AT. Penerapan Teknik Bagging Pada Algoritma Naive Bayes Dan Algoritma C4.5 Untuk Mengatasi Ketidakseimbangan Kelas. J Bisnis Digit dan Sist Inf. 2020;1:41–48.
21. Brownlee J. Ensemble Learning Algorithms With Python: Make Better Predictions with Bagging, Boosting, and Stacking [Internet]. Machine Learning Mastery; 2021. 450 p. Available from: https://books.google.co.id/books?id=IUkrEAAAQBAJ
22. Widyaningsih Y, Arum GP, Prawira K. Aplikasi K-Fold Cross Validation Dalam Penentuan Model Regresi Binomial Negatif Terbaik. BAREKENG: Jurnal Ilmu Matematika dan Terapan. 2021 Jun 1;15(2):315-22.Available from: https://doi.org/10.30598/barekengvol15iss2pp315-322