PREDICTING STROKE RISK: A MULTI-ALGORITHM APPROACH USING MACHINE LEARNING

Satya Kiran Kota Akshit Sharma
Sai Srinivas Munagala Nitish Kumar Pinneti

Abstract

This final report presents the culmination of our project's efforts to improve cerebral stroke prediction using machine learning. Beginning with sophisticated exploratory data analysis, we uncovered key patterns and relationships within the dataset sourced from Kaggle. However, the crux of our project is the exhaustive implementation of various classification algorithms, including as Logistic Regression, Decision Trees, Random Forests, and Voting Classifiers, which were primarily designed from scratch. We obtained significant findings after rigorous testing and assessment, these findings highlight the effectiveness of our technique, providing vital insights for enhancing stroke prediction approaches and improving patient outcomes in healthcare facilities.

1. Introduction

Stroke prediction poses a critical challenge in healthcare due to its potential for severe consequences. To overcome this, machine learning (ML) approaches are used, harnessing physiological characteristics to make accurate predictions. However, the imbalance in stroke datasets needs the adoption of oversampling approaches to assure reliable model training.

To address dataset imbalance, this research effort uses both the Synthetic Minority Over-sampling Technique (SMOTE) and Random Oversampling. SMOTE creates synthetic samples to balance class distribution, whereas random oversampling replicates minority class occurrences to provide a balanced dataset. These strategies improve the dependability of model predictions by ensuring that stroke instances are adequately represented.

Furthermore, this report highlights the use of ML techniques mostly from scratch, emphasizing the need of rigor and control over model construction. Wherever possible, algorithms like Logistic Regression (LR), Decision Tree (DT) Classification, and Voting Classifiers are implemented independently. However, in cases where custom implementation is impractical, library calls are made to established ML libraries for efficient algorithm execution.

This comprehensive approach, integrating oversampling techniques with custom ML algorithm implementation, results in efficient stroke prediction models. We hope the research makes a substantial contribution to the improvement of stroke prediction approaches by combining rigorous scientific evidence with practical application, providing vital insights for both healthcare practitioners and academics.

2. Methodology

This section outlines the approach taken for data cleaning, exploratory analysis and preprocessing.

2.1. Data Cleaning and Null-Value Handling

The dataset utilized for this project comprised cerebral stroke prediction details. It contained *43,400* entries across *12* columns, with no identified duplicate records and null values in *bmi* and *smoking status*.

Table 1. Features with NULL values.

COLUMN NAME	DATA TYPE	NULL COUNT
BMI	$float64$	1462
SMOKING_STATUS	$object$	13292

Handling null values was crucial without compromising dataset integrity. For *bmi*, median imputation was applied due to detected outliers, while *smoking status* null values were replaced with the *unknown* category, introducing a new data dimension. Post-null value mitigation, redundant columns were identified and removed. The 'id' column, lacking predictive significance, was eliminated, while all other columns were retained.

2.2. Exploratory Data Analysis

Exploratory data analysis, including plotting column occurrences, aided in understanding dependencies and stroke correlations. During exploratory data analysis (EDA), we

examined the distribution of each feature with respect to the stroke value, analyzing the occurrence of each value within each column. Additionally, we investigated the dependencies between two features by utilizing correlation and covariance matrices. This allowed us to gain insights into potential patterns, class imbalances, anomalies, and associations within the data.

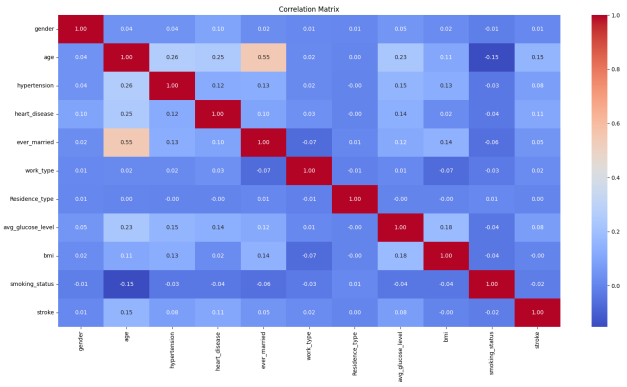

Figure 1. EDA: The correlation matrix of the columns.

2.3. Preprocessing

During the midterm stage of the project, preprocessing of the dataset involved several steps to prepare the data for model training and evaluation.

2.3.1. Encoding and Splitting

Label encoding was applied to convert categorical variables into numerical format, facilitating compatibility with machine learning algorithms. However, it was noted that label encoding introduces ordinality, which may not always be appropriate for categorical variables with more than two categories.

Then the dataset was partitioned into distinct sections for training and testing to ensure unbiased model evaluation and performance assessment. Data shuffling was performed to evenly distribute samples, with *25%* of the data reserved for testing purposes. The remaining dataset was allocated for training ML models. The random state for shuffling was set to *42*.

2.3.2. Handling Data Imbalance

To address class imbalance observed in the training dataset, various sampling techniques were explored. Given the significant disparity in class distribution (*31975* instances of non-stroke (0) and *575* instances of stroke (1)), under sampling methods risked data loss.

Therefore, over sampling methods, including Random Over Sampling and Synthetic Minority Over-sampling Technique (*SMOTE*), were considered more suitable. Their implementaiton details in the following sections.

Algorithm 1 Random Over Sampling

Require: X: Feature matrix, y: Labels, minority class label
1: Identify indices of majority and minority class samples
2: Sample minority class indices with replacement to match majority class size
3: Combine minority class samples with majority class samples
4: Shuffle combined indices
5: Create resampled feature matrix and target labels
Ensure: $X_{\text{resampled}}$, $y_{\text{resampled}}$

2.4. Sampling Techniques

2.4.1. Random Over Sampling (ROS)

Instances of the minority class (stroke) were randomly replicated to balance the class distribution. This technique mitigates bias and improves model performance by providing more balanced training data.

2.4.2. Synthetic Minority Over-sampling Technique (SMOTE)

SMOTE generates synthetic samples for the minority class based on feature space similarity between existing minority samples. This method creates synthetic data points resembling minority class instances, addressing class imbalance without replicating existing samples.

Algorithm 2 SMOTE

Require: X: Features, y: Labels, k: Number of neighbors, $ratio$: Ratio of synthetic samples
Calculate class distribution
Find minority and majority classes
Initialize $X_{\text{resampled}}$, $y_{\text{resampled}}$
Fit k-nearest neighbors
for i **in** range(length(X)) **do**
  if $y[i]$ is minority class **then**
    Find k-nearest neighbors
    for j **in** range(int($ratio$)) **do**
      Generate synthetic sample and append
    end for
  end if
  if $y[i]$ is majority class **then**
    Append sample
  end if
end for
Ensure: $X_{\text{resampled}}$, $y_{\text{resampled}}$

These sampling techniques were employed to handle class imbalance in the dataset before applying machine learning algorithms. They play a crucial role in improving the effectiveness of the predictive models.

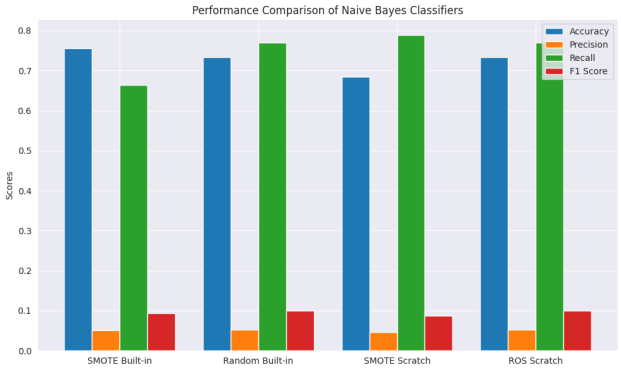

Figure 2. Confusion Matrix of Hard Voting Classifier.

3. Algorithms and Implementation

In this section, we detail the implementation and evaluation of various machine learning algorithms for stroke prediction.

3.1. Logistic Regression

Logistic regression was employed as one of the initial algorithms to predict the likelihood of stroke based on various demographic, lifestyle, and medical factors.

3.1.1. Input and Output Representation

Model Inputs: The dataset consisted of n rows and m columns, denoted as X, where X is an $n \times m$ matrix representing the input features.

Actual Output: Denoted as $\underline{y}$, it was an $n \times 1$ vector containing the true labels for each data point.

Predicted Output: Denoted as $\underline{\hat{y}}$, it was also an $n \times 1$ vector representing the model's predictions.

Weights and Bias: The logistic regression model utilized weights ($\underline{\theta}$) of size $m \times 1$ and a bias term (b).

Learning rate: The logistic regression uses η as learning rate, it's a hyper parameter.

Algorithm 3 Logistic Regression

Initialize: $\theta_j = 0$ for all $0 \leq j \leq m$, size m
repeat
$\underline{\hat{y}} = sigmoid(X \cdot \underline{\theta} + \underline{b})$
$d\theta = \frac{1}{n}[X^T \cdot (w_1 \underline{y}(1 - \underline{\hat{y}}) + w_0(1 - \underline{y})\underline{\hat{y}})]$
$db = \sum_{i=1}^{n}(w_1 \underline{y}(1 - \underline{\hat{y}}) + w_0(1 - \underline{y})\underline{\hat{y}})$
$\theta = \theta - \eta d\theta$
$b = b - \eta db$
until k *iterations*

3.1.2. Loss Function and Optimization

Initially, the model underwent training using gradient descent optimization to minimize the *standard binary cross-entropy loss* function.

Upon observation of Fig. 2, it became apparent that the accuracy reached 98%. However, in the context of an imbalanced dataset, accuracy alone does not adequately capture model performance. Notably, the confusion matrix depicted in Fig. 2 illustrates that all instances are predicted as 0s, highlighting the limitations of accuracy as a metric.

Given the dataset's class imbalance, we subsequently opted for a weighted loss function to better address this issue.

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

The *weighted loss function* was introduced to address this imbalance, where the number of instances of the negative class (0s) was significantly higher than the positive class (1s).

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}[w_0(y_i \log(\hat{y}_i)) + w_1((1-y_i)\log(1-\hat{y}_i))]$$

Formula for class weights: $w_j = \frac{n_{samples}}{(n_{classes} \times n_{samples_j})}$

3.1.3. Learning Rate

A lower learning rate leads to slower training times, while a higher learning rate can cause oscillations in the loss value. Therefore, it is crucial to find an optimal learning rate that balances faster training rates with better performance. After experimenting with various learning rates, we determined that a learning rate of 1 is optimal. A learning rate of 10 resulted in oscillations, while a rate of 0.001 required more iterations to train the model.

3.1.4. Regularization

After conducting experiments with L1, L2, and no regularization, we observed that there was no significant change in

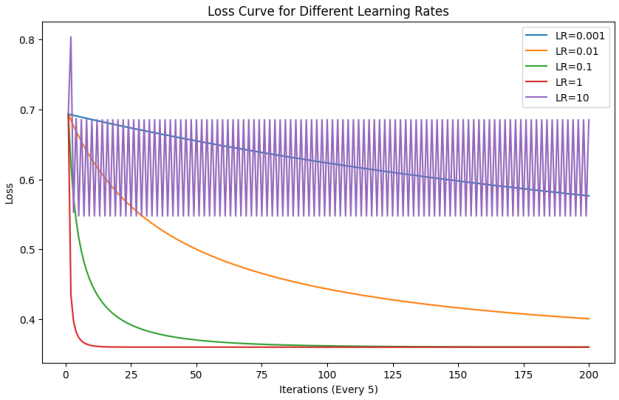

Figure 3. Loss curve at different learning rates.

the performance of the model during testing. Consequently, we decided not to employ any form of regularization.

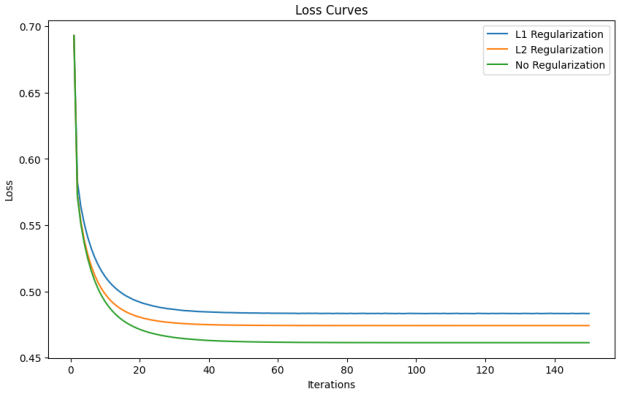

Figure 4. Regularization vs Loss Curve.

3.1.5. Threshold Adjustment for Improved Performance

Given the importance of recall in stroke detection, manual thresholding was performed to optimize recall.

Manual Thresholding: This involved adjusting the threshold value for class prediction to achieve a higher recall without compromising overall accuracy.

Threshold Optimization: Through manual thresholding, a threshold of 0.0061 was identified to yield a balance between recall and accuracy. Iterative adjustment and validation on the test set were performed to determine the optimal threshold.

After applying manual thresholding and employing the weighted loss function, the final confusion matrix, presented

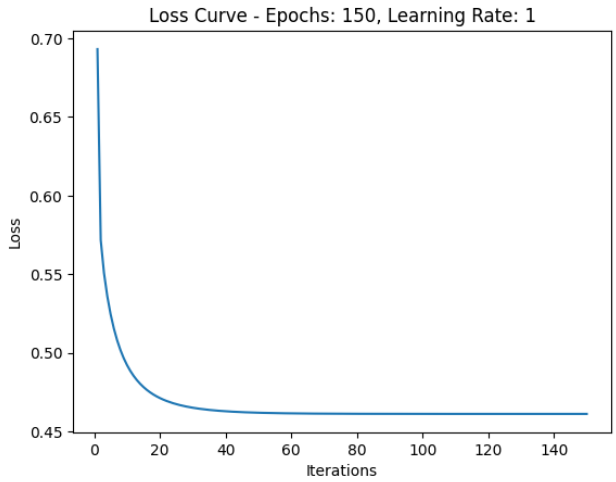

Figure 5. MSE vs iterations.

in Fig. 7, was obtained. Despite exhibiting lower accuracy compared to the model using the actual loss function, this approach proves superior in stroke prediction. Notably, it successfully classifies some instances of 1s, enhancing its utility for this task. Additionally, the comparison with the logistic regression model from the scikit-learn library revealed similar outcomes, thus validating the efficacy of our logistic regression implementation.

3.2. Decision Tree

The dataset was also evaluated using decision tree- which was implemented from the scratch.

3.2.1. Input and Output Representation

Model Inputs: The dataset consisted of n rows and m columns, denoted as X, where X is an $n \times m$ matrix representing the input features.

Actual Output: Denoted as $\underline{y}$, it was an $n \times 1$ vector containing the true labels for each data point.

Predicted Output: Denoted as $\underline{\hat{y}}$, it was also an $n \times 1$ vector representing the model's predictions.

Tree Structure: The Decision Tree model utilizes a tree-like model of decisions. The tree is built by splitting the dataset into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning.

Split Criterion: The Decision Tree uses a metric to decide which attribute to split on at each step in building the tree. Common metrics are Information Gain, Gain Ratio, or Gini Impurity.

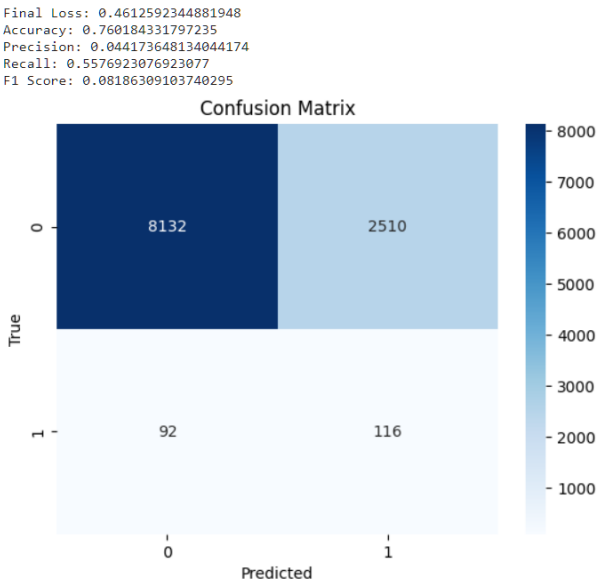

Figure 6. Confusion Matrix for weighted loss function with optimal threshold.

Algorithm 4 Decision Tree

Initialize: $N =$ root node
repeat
  $A = \arg\max_a$ Information Gain(a)
  $D_1, D_2, ..., D_k =$ split(D, A)
  $N =$ node(A)
  for $i = 1$ to k **do**
    Recurse on D_i with $A - \{A\}$
  end for
  Return N
until $\forall d \in D$, class$(d) = c$ or $A = \emptyset$

Tree Structure: The Decision Tree model utilizes a tree-like model of decisions. The tree is built by splitting the dataset into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning.

Split Criterion: The Decision Tree uses a metric to decide which attribute to split on at each step in building the tree. Common metrics are Information Gain, Gain Ratio, or Gini Impurity.

3.2.2. Training of Decision Tree

Decision tree was trained on all the three types of datasets with a max depth of 4 and minimum number of sample leaves as 1-

1. Dataset processed via Random Oversampling
2. Dataset processed via SMOTE technique
3. Original biased dataset

It was then evaluated on the testing dataset, whose results have been summarized in figure 8

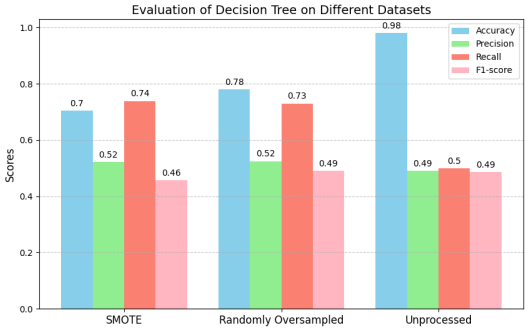

Figure 7. Evaluation of decision tree on different types of datasets

Of the results obtained, the result obtained on the Original dataset was neglected considering the presence of high bias to the absence of stroke (31975:575). Of the remaining two, Dataset processed via Random Oversampling to eliminate bias gave better results- thus all the further evaluation is performed on Randomly Oversampled dataset.

3.2.3. Variation of Decision Tree Performance with Depth

A decision tree was trained on the randomly oversampled dataset for different values of maximum depth keeping the minimum number of leaves per node to be 1. The results obtained are shown in figure **??**

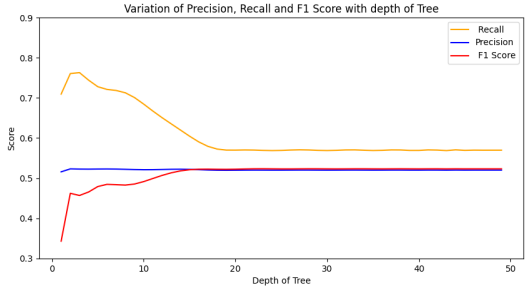

Figure 8. Variation of Decision Tree Performance with depth of tree

A maximum of 0.523 F1 score was obtained at a depth of 20, while a maximum of recall and precision of 0.76,0.52 were obtained at a depth of 1,4 respectively. Since F1 takes into account both precision and recall, we consider it to be a better metric and perform further evaluation on a max tree depth of 20

3.2.4. Variation of Decision Tree Performance with Min. Number of Leafs in A Node

Next, variation of decision tree performance with the minimum number of leafs in a node is performed, taking the maximum tree depth to be 20.

A maximum of 0.523 F1 score was obtained at a depth of 20, while a maximum of recall and precision of 0.76,0.52 were obtained at a depth of 1,4 respectively. Since F1 takes into account both precision and recall, we consider it to be a better metric and perform further evaluation on a max tree depth of 20

3.3. Naive Bayes

The Naive Bayes classifier was also employed in the initial stage of predicting stroke likelihood, leveraging various demographic, lifestyle, and medical factors.

3.3.1. Input and Output Representation

Model Inputs: The dataset comprises n rows and m columns, denoted as X, where X is an $n \times m$ matrix representing the input features.

Actual Output: Denoted as $\underline{y}$, it is an $n \times 1$ vector containing the true labels for each data point, representing the occurrence or absence of stroke.

Predicted Output: Denoted as $\underline{\hat{y}}$, it is also an $n \times 1$ vector representing the model's predictions of stroke likelihood for each data point.

3.3.2. Algorithm

The Naive Bayes classifier, based on Bayes' theorem, assumes independence among features. It calculates the probability of each class given the input features and selects the class with the highest probability as the prediction.

3.3.3. Performance and Model Evaluation

Naive Bayes exhibits efficient performance in predicting stroke likelihood. With its assumption of feature independence, it provides a computationally lightweight solution for classification tasks. However, its simplistic nature may lead to suboptimal performance in capturing complex relationships within the data, compared to more advanced models like logistic regression.

Algorithm 5 Naive Bayes Classifier

Training:
Calculate class priors: $P(y_i) = \frac{\text{count}(y_i)}{n}$
Calculate class-conditional probabilities: $P(x_j|y_i)$ for all features x_j
Testing:
for each data point x in test set **do**
  Calculate $P(y_i|x)$ for all classes using Bayes' theorem
  Select the class with the highest probability as the predicted class
end for

3.3.4. Handling Imbalanced Data

Naive Bayes, like logistic regression, may face challenges with imbalanced datasets. While it does not directly address this issue during training, post-training techniques such as adjusting decision thresholds or using weighted loss functions can be applied to improve its performance on imbalanced data.

```
Accuracy: 0.7331797235023041
Precision: 0.05320917858330562
Recall: 0.7692307692307693
F1-score: 0.09953343701399689
Confusion Matrix:
```

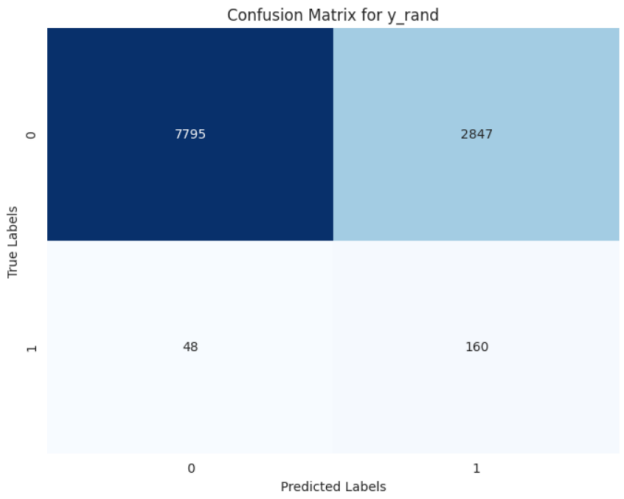

Figure 9. Confusion Matrix of Naive Bayes.

3.3.5. Threshold Adjustment for Improved Performance

Threshold adjustment involves setting a threshold on the class probability estimates to make predictions. By adjusting this threshold, Naive Bayes predictions can be optimized to prioritize specific metrics such as recall or precision. This fine-tuning can significantly enhance the model's perfor-

mance in scenarios like stroke prediction.

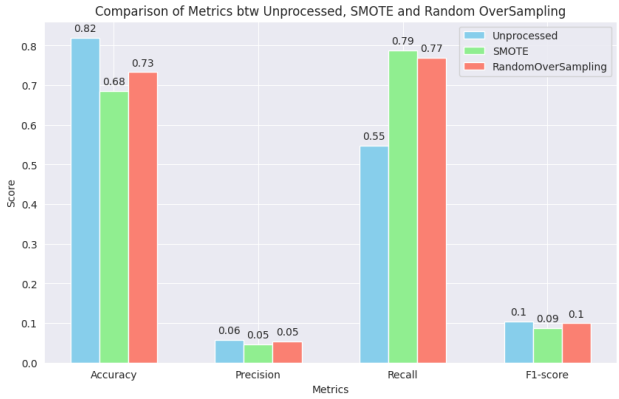

Figure 10. Metrics Comparison for Naive Bayes.

3.3.6. Comparison with Logistic Regression

Comparing Naive Bayes with logistic regression, the former offers simplicity and computational efficiency due to its assumption of feature independence. However, logistic regression can capture more complex relationships in the data, potentially leading to better predictive performance in certain scenarios. Experimentation and validation on the dataset are essential for determining which classifier performs better for a specific task, such as predicting stroke likelihood.

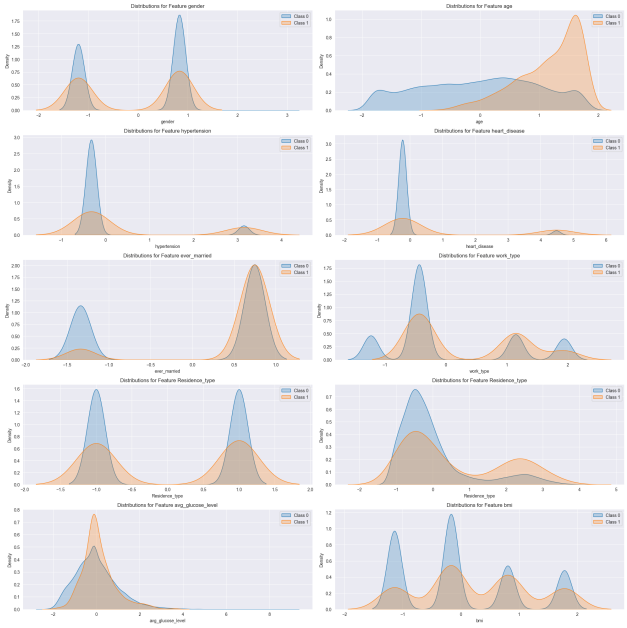

Figure 11. Naive Bayes: Feature Distributions.

Algorithm 6 Hard Voting Classifier

Require: Trained individual classifiers: $C_1, C_2, ..., C_n$
Require: Test dataset: X_{test}
Ensure: Predicted labels: $\hat{y}$
  for each data point x in X_{test} **do**
    for each classifier C_i **do**
      Predict label $\hat{y}_i$ using C_i
    end for
    Count votes for each class based on $\hat{y}_1, \hat{y}_2, ..., \hat{y}_n$
    Select the class with the majority of votes as the final prediction $\hat{y}$
  end for

4. Voting Classifier with Hard Voting

The voting classifier combines the predictions of multiple individual classifiers to make a final prediction. In the case of hard voting, the majority vote among the classifiers determines the final prediction.

4.1. Input and Output Representation

Model Inputs: The dataset consists of n rows and m columns, denoted as X, where X is an $n \times m$ matrix representing the input features.

Actual Output: Denoted as $\underline{y}$, it is an $n \times 1$ vector containing the true labels for each data point.

Predicted Output: Denoted as $\underline{\hat{y}}$, it is also an $n \times 1$ vector representing the ensemble model's predictions.

4.2. Algorithms Used

The voting classifier utilizes the following algorithms:

- Logistic Regression
- Naive Bayes
- Decision Tree

4.2.1. Training Process

Each individual classifier (logistic regression, Naive Bayes, and decision tree) is trained on the dataset independently.

4.2.2. Voting Mechanism

During prediction, each classifier generates its prediction. The final prediction is determined by a majority vote among the classifiers.

4.3. Performance and Model Evaluation

The voting classifier with hard voting combines the strengths of multiple classifiers to improve overall performance. By

combining the predictions of individual classifiers, it can achieve better generalization and accuracy compared to any single classifier alone.

4.3.1. Model Ensemble

The ensemble of classifiers works together to make predictions. By considering the collective decision of multiple classifiers, the voting classifier reduces the risk of overfitting and increases model stability.

4.3.2. Decision Boundary

The decision boundary of the voting classifier is influenced by the decision boundaries of the individual classifiers. Through majority voting, the voting classifier may create a more complex decision boundary compared to individual classifiers.

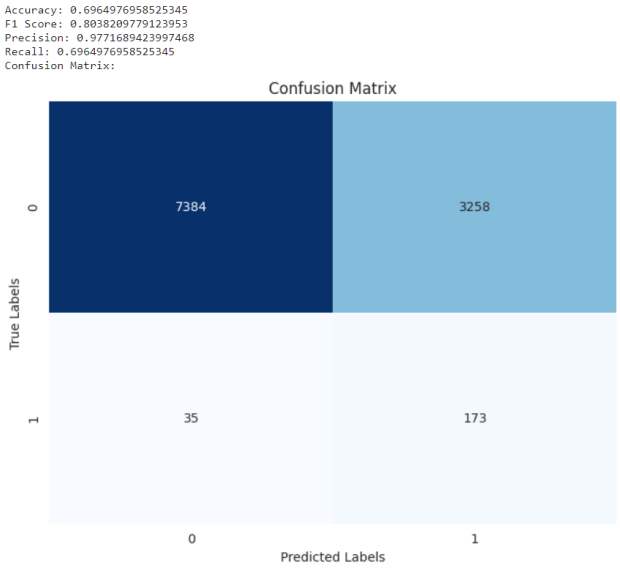

Figure 12. Confusion Matrix of Hard Voting Classifier.

4.4. Handling Imbalanced Data

Similar to individual classifiers, the voting classifier may encounter challenges with imbalanced datasets. Post-training techniques such as adjusting decision thresholds or using weighted voting can be employed to address imbalanced data.

4.5. Threshold Adjustment for Improved Performance

Threshold adjustment involves setting a threshold on the probability estimates from each classifier to make predictions. By adjusting these thresholds, the voting classifier's predictions can be optimized to prioritize specific metrics such as recall or precision.

4.6. Comparison with Individual Classifiers

Compared to individual classifiers such as logistic regression, Naive Bayes, and decision trees, the voting classifier offers improved performance through ensemble learning. By combining the strengths of multiple classifiers, it can achieve better predictive accuracy across diverse datasets.

5. Conclusion

The project delved into a multi-algorithm approach for predicting stroke risk using machine learning techniques. Managing the inherent class imbalance in the dataset emerged as a critical challenge, wherein oversampling methods such as random oversampling and SMOTE proved effective. Logistic regression, coupled with a weighted loss function and manual threshold adjustment, achieved a promising recall of 0.807 for stroke prediction.

Table 2. Model Evaluation Metrics.

Model	F1	Accuracy	Precision	Recall
LR(S)	0.081	0.749	0.043	0.576
LR	0.104	00.735	0.056	0.807
NB(S)	0.099	0.733	0.053	0.769
NB	0.10	0.733	0.050	0.77
DT(S)	0.523	0.78	0.52	0.76
HV	0.095	0.696	0.050	0.831

* Models with (S) were implemented from scratch.

The decision tree algorithm, implemented from scratch, attained its optimal F1 score of 0.523 at a maximum depth of 20 on the randomly oversampled data. While naive Bayes offered a computationally efficient solution, its performance fell behind logistic regression. Notably, the hard voting classifier, an ensemble method combining logistic regression, naive Bayes, and decision trees, showcased superior overall performance with a high recall of 0.831, leveraging the strengths of multiple models.

Techniques like threshold adjustment and model ensembling proved invaluable in optimizing crucial metrics like recall for the stroke prediction task. This comprehensive approach, integrating custom algorithm implementation, data preprocessing, and ensemble methods, yielded valuable insights into enhancing stroke prediction models and improving healthcare outcomes.

References

Dritsas, E. and Trigka, M. Stroke risk prediction with machine learning techniques. *Sensors (Basel)*, 22(13):4670, 2022. doi: 10.3390/s22134670.

Satya, Akshit, Nitish, and Srinivas. Predicting stroke risk data cleaning. https://www.kaggle.com/code/satya514/predicting-stroke-risk-data-cleaning, 2021.