Binary Classification with a Tabular Stroke Prediction Dataset

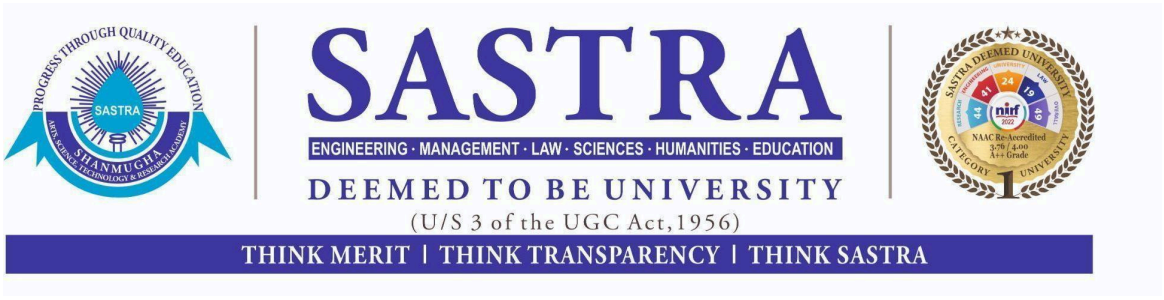
Report submitted to SASTRA Deemed to be
University As per the requirement for the course

CSE425 : MACHINE LEARNING ESSENTIALS

Submitted by

SABBINA SRI HARSHA
(Reg No: 125018056, B. Tech Computer Science and Business Systems)

OCTOBER- 2024



SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

Table of Contents

ABSTRACT

This project aims to predict the risk of stroke using various machine learning models based on health-related features such as age, hypertension, heart disease, smoking status, and BMI. The dataset comprises over 5,000 records and 12 features. Models such as Logistic Regression, Random Forest, XGBoost, and Support Vector Machines were applied for binary classification of stroke risk. The models were evaluated based on accuracy, precision, recall, and F1-score. XGBoost emerged as the most accurate model after hyperparameter optimization, with a final accuracy of 96%.

INTRODUCTION

Stroke is one of the leading causes of death and disability worldwide. Timely prediction of stroke risk can help in reducing mortality and ensuring better healthcare outcomes through early intervention. This project focuses on predicting the likelihood of a stroke using machine learning techniques by analyzing individual health records.

Objective:

The primary objective of this project is to develop a machine learning model that can predict stroke risk with high accuracy, allowing healthcare providers to intervene early and manage patient health more effectively.

Related Work

Several machine learning studies have focused on predicting cardiovascular events, including stroke. Techniques like Random Forest, Logistic Regression, and Support Vector Machines have been employed to predict stroke based on risk factors such as age, hypertension, and lifestyle habits. Past work has highlighted challenges in handling class imbalance, as stroke occurrences are rare in datasets, making accurate prediction difficult without appropriate handling of minority classes.

Approach: Use classification models with a focus on predicting the obesity level of individual by comparison the results based on 8 models.

Background

Key Machine Learning Models:

Logistic Regression:Logistic Regression is a linear model used for binary classification. Despite its name, it is used to model the probability of a binary outcome, mapping input features to a 0 or 1 class. The output of logistic regression is the probability of the instance belonging to the positive class (stroke). If the probability is above a certain threshold (typically 0.5), the instance is classified as positive (stroke), otherwise, it is classified as negative (no stroke).

Random Forest:Random Forest is an ensemble learning method that constructs a "forest" of decision trees during training. Each tree is trained on a random subset of the data, and its predictions are combined (via averaging or majority voting) to produce the final prediction. The randomness injected into the model helps improve generalization and reduces the risk of overfitting, which is common in single decision trees.

XGBoost:XGBoost is an advanced implementation of gradient boosting algorithms, specifically designed for speed and performance. It builds an ensemble of decision trees sequentially, with each new tree trying to correct the errors made by previous ones. XGBoost is highly effective due to its use of advanced regularization (to prevent overfitting) and its ability to handle large-scale datasets.

Support Vector Machines (SVM): SVM is a powerful classification algorithm that works by finding the hyperplane that best separates data points into two classes (stroke vs. no stroke). The goal is to maximize the margin—the distance between the hyperplane and the nearest data points (support vectors) from each class.

K-Nearest Neighbors (KNN):K-Nearest Neighbors (KNN) is a simple and intuitive algorithm used for both classification and regression tasks. It classifies a data point based on the labels of its nearest neighbors in the feature space. The principle behind KNN is that data points that are close to each other are more likely to belong to the same class.

Decision Tree:A **Decision Tree** is a non-linear algorithm used for both classification and regression tasks. It builds a tree-like structure where each internal node represents a decision based on a feature, each branch represents the outcome of the decision, and each leaf node represents the final prediction (class or value).

Gaussian Naive Bayes:Naive Bayes is a family of simple "probabilistic classifiers" based on Bayes' Theorem, with the assumption that the features are conditionally independent given the class label. **Gaussian Naive Bayes** assumes that the continuous features follow a normal (Gaussian) distribution.

Gradient Boosting:Gradient Boosting is an ensemble learning technique used for both classification and regression tasks. It builds a model in a stage-wise fashion by sequentially adding models (usually decision trees) that correct the errors made by previous models.

Evaluation Metrics:

- **Accuracy**: Measures the overall correctness of the model.
- **Precision and Recall**: Helps assess the trade-off between false positives and false negatives.
- **F1-Score**: Balances precision and recall, useful for imbalanced datasets like stroke prediction.

MATH BEHIND THE MODELS:

1. Logistic Regression for Classification:

Logistic Regression is a supervised learning algorithm used for binary classification problems. It models the probability of a binary outcome as a linear combination of input features, but instead of a linear relationship, it uses a logistic (sigmoid) function to map the output between 0 and 1, representing the probability of class membership.

Step 1: Hypothesis

In logistic regression, the hypothesis is based on a **linear combination** of the input features X, but the output is mapped through the **sigmoid function** to constrain it between 0 and 1. The hypothesis function is defined as:

$$h_\theta(X) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 X_1 + \theta_2 X_2 + \cdots + \theta_n X_n)}}$$

\

Here:

- $h_\theta(X)$ is the predicted probability that the output Y belongs to class 1 (e.g., $P(Y = 1|X)$),
- $\theta_0, \theta_1, \ldots, \theta_n$ are the model parameters (weights),
- $X_1, X_2, \ldots, X_n$ are the input features.

Step 2: Sigmoid (Logistic) Function

The sigmoid function ensures the predicted output is always between 0 and 1:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

where $z = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \cdots + \theta_n X_n$. This function outputs values close to 0 for large negative z, and values close to 1 for large positive z.

Step 3: Decision Boundary

The predicted probability hθ(X) is then compared to a **threshold** (typically 0.5) to classify the outcome:

- If $h_\theta(X) \geq 0.5$, predict $Y = 1$,
- If $h_\theta(X) < 0.5$, predict $Y = 0$.

This threshold creates a decision boundary in the feature space where the classification changes from 0 to 1.

Step 4: Cost Function (Log-Loss)

Instead of minimizing the sum of squared errors (as in linear regression), logistic regression uses a **logistic loss function** (also known as log-loss or binary cross-entropy) to measure the error:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right]$$

Here:

- m is the number of training examples,
- $y^{(i)}$ is the actual label for the i-th training example,
- $h_\theta(x^{(i)})$ is the predicted probability for the i-th example.

Step 5: Optimization (Gradient Descent)

To minimize the cost function and find the best parameters θ, **Gradient Descent** is used. The update rule for gradient descent is:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

for each parameter θ_j, where:

- α is the learning rate (a small positive value that controls the step size),
- $\frac{\partial}{\partial \theta_j} J(\theta)$ is the derivative of the cost function with respect to θ_j.

The partial derivative of the cost function for logistic regression is:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Step 6: Regularization

To prevent overfitting, especially when dealing with high-dimensional data, **regularization** can be applied. A regularization term is added to the cost

function to penalize large weights. The regularized cost function (for L2 regularization) becomes:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

where λ is the regularization parameter that controls the strength of the penalty.

2. Random Forest Classification:

Random Forest is an ensemble method that combines multiple decision trees. It improves accuracy by reducing overfitting through random sampling of both data and features.

Step 1: Bootstrap Sampling

- Multiple subsets of the training data are generated by randomly sampling with replacement (bootstrapping).

Step 2: Tree Construction

- For each subset, a decision tree is built using a random subset of features. The tree is split based on a criterion such as Gini impurity or Entropy.
- **Gini Impurity:**

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

where pi is the proportion of samples in class i, and C is the number of classes.

Entropy (used in information gain):

$$H(D) = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

The goal is to maximize the information gain or reduce impurity at each split.

3. Gaussian Naive Bayes Classification:

Naive Bayes is based on **Bayes' Theorem** and assumes that all features are independent given the class label. Gaussian Naive Bayes is a variant where the likelihood of the features is assumed to follow a Gaussian (normal) distribution.

Step 1: Bayes' Theorem

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

- P(y | X) is the posterior probability of class y given input X, P(X | y) is the likelihood, P(y) is the prior, and P(X) is the evidence.

Step 2: Gaussian Likelihood

- For Gaussian Naive Bayes, the likelihood P(Xi | y)P for each feature is modeled as a Gaussian distribution:

$$P(X_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(X_i - \mu_y)^2}{2\sigma_y^2}\right)$$

- Here, μy and σ^2y are the mean and variance of the feature Xi for class y.

Step 3: Prediction

- The class with the highest posterior probability is selected as the predicted class

$$y = \arg\max P(y|X)$$

4. Support Vector Classification:

Support Vector Machines (SVM) aim to find the hyperplane that best separates the data into classes while maximizing the margin between the classes.

Step 1: Objective Function

- SVC tries to solve the optimization problem:

$$\min \frac{1}{2}||w||^2$$

subject to:

$$y_i(w \cdot x_i + b) \geq 1 \quad \forall i$$

Here, w is the weight vector, b is the bias term, and yi are the class labels. The goal is to find w and b such that the margin between the two classes is maximized.

Step 2: Support Vectors

- The points that lie on the edge of the margin are called **support vectors**. These points define the decision boundary.

Step 3: Kernel Trick (Non-linear data)

- For non-linearly separable data, SVM uses a **kernel trick** to map the data

into a higher-dimensional space. A common kernel is the **Radial Basis Function (RBF)** kernel:

$$K(x, x') = \exp\left(-\gamma ||x - x'||^2\right)$$

5. XGBoost Classification

XGBoost is a powerful implementation of gradient boosting that builds models sequentially, with each new model correcting the errors of the previous ones.

Step 1: Initialization

- Start with an initial prediction, usually the mean of the target variable.

Step 2: Gradient Descent

- At each iteration, compute the residuals (errors) of the current model and fit a new weak learner (typically a decision tree) to these residuals.
- The objective is to minimize the loss function L:

$$L = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

- Here, l is the loss function (e.g., log-loss for classification), and Ω(fk) is a regularization term that penalizes model complexity.

Step 3: Update Predictions

- The predictions are updated by adding the output of the new weak learner to the previous prediction:

$$\hat{y}_i^{(t+1)} = \hat{y}_i^{(t)} + \eta f_t(x_i)$$

- Here, η\etaη is the learning rate, and ft(xi)f_t(x_i)ft(xi) is the prediction of the new weak learner.

6. Gradient Boosting Classification:

Gradient Boosting also builds models sequentially, but it optimizes a differentiable loss function by training weak learners to correct the errors made by previous models.

Step 1: Initialization

- Start by predicting the mean value of the target variable for all instances.

Step 2: Residual Calculation

- Compute the residuals (differences between the predicted and actual values) from the current model:

$$r_i = y_i - \hat{y}_i$$

Step 3: Fit Weak Learners

- Train a weak learner (usually a decision tree) to fit the residuals.

Step 4: Update Prediction

- Update the model by adding the predictions of the weak learner to the current model's predictions

$$\hat{y}_i^{(t+1)} = \hat{y}_i^{(t)} + \eta f_t(x_i)$$

- Here, ft(xi) is the new weak learner's prediction, and η is the learning rate.

Step 5: Repeat

- Repeat steps 2-4 until a stopping criterion is met, such as a maximum number of iterations or when the improvement in the loss function becomes negligible.

DATASET DESCRIPTION

The dataset used for this project was sourced from the Kaggle Playground Stroke Prediction Competition. It contains over 5,000 records and includes the following features:

- **id:** Unique identifier for each patient.
- **gender:** Categorical variable representing male, female, or other.
- **age:** Numeric variable representing the patient's age.
- **hypertension:** Binary variable indicating if the patient has hypertension (1) or not (0).
- **heart_disease:** Binary variable indicating if the patient has heart disease (1) or not (0).
- **ever_married:** Categorical variable indicating if the patient has ever been married (Yes/No).
- **work_type:** Categorical variable representing the patient's employment type.
- **Residence_type:** Categorical variable indicating urban or rural

residence.

- **avg_glucose_level:** Numeric variable representing the patient's average glucose level.
- **bmi:** Numeric variable representing the patient's body mass index.
- **smoking_status:** Categorical variable indicating the patient's smoking history.

Data Sample :

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	0	Male	28.0	0	0	Yes	Private	Urban	79.53	31.1	never smoked	0
1	1	Male	33.0	0	0	Yes	Private	Rural	78.44	23.9	formerly smoked	0
2	2	Female	42.0	0	0	Yes	Private	Rural	103.00	40.3	Unknown	0
3	3	Male	56.0	0	0	Yes	Private	Urban	64.87	28.8	never smoked	0
4	4	Female	24.0	0	0	No	Private	Rural	73.36	28.8	never smoked	0

- **Preprocessing**:

  Preprocessing done on the data:

  Handling Missing Values: BMI had some missing values, which were imputed using the mean value.

  standard practice to verify and handle any potential missing or null entries.

  Encoding Categorical Features: Categorical variables (e.g., gender, smoking status) were encoded using one-hot encoding. transformed into numerical representations using LabelEncoder.

  Feature Scaling: Continuous variables such as age, glucose level, and BMI

were scaled using standardization for better performance in machine learning models.

Data Splitting: The preprocessed data is divided into training and testing sets to evaluate the performance of machine learning models (80% for training data and 20% for testing the data).

METHODOLOGY:

Preprocessing Steps:

1. **Missing Data Imputation**: Missing BMI values were filled using the dataset's average BMI.
2. **One-Hot Encoding**: Categorical features were encoded to numerical formats (e.g., male/female, smoking status).
3. **Standardization**: Continuous features were scaled to have a mean of 0 and a standard deviation of 1.

Data Splitting:

The dataset was divided into 80% training data and 20% test data to

Results

Each of the models have been evaluated for their correctness based on the below metric.

1. **Precision:**
    - Definition: Precision indicates how many of the predicted positive cases were actually positive.
    - Formula: Precision=True Positives/(True Positives + False Positives)
    - Use Case: High precision is crucial in scenarios where the cost of false positives is high (e.g., diagnosing a rare disease).

2.Recall (Sensitivity):

- Definition: Recall shows how many actual positive cases were identified by the model.
- Formula: Recall=True Positives/(True Positives + False Negatives)
- Use Case: High recall is essential in cases where missing a positive instance is critical (e.g., detecting fraud).

3. F1 Score:

- Definition: The F1 score is the harmonic mean of precision and recall, providing a single metric that balances both concerns.
- Formula: F1 Score=2((Precision · Recall)/(Precision + Recall)).
- Use Case: Useful when you need a balance between precision and recall, especially in imbalanced datasets.

4. Support:

- Definition: This indicates the number of actual occurrences of each class in the specified dataset.
- Use Case: It helps to understand how many instances there are for each class, which is useful for evaluating metrics in relation to the size of each class.

Result Visualization:

The below is the result obtained from the implementation of Logistic Regression

```
Accuracy is:  0.9621038876184254
```

Below is the result obtained from the implementation of XGBoost Classification.

```
Accuracy is:  0.9617771969944463
```

Below is the result obtained from the implementation of Decision Tree.

```
Accuracy is:  0.9624305782424044
```

Below is the result obtained from the implementation of Random Forest Classification.

	precision	recall	f1-score	support
0	0.96	1.00	0.98	2931
1	0.40	0.06	0.11	130
accuracy			0.96	3061
macro avg	0.68	0.53	0.54	3061
weighted avg	0.94	0.96	0.94	3061

Below is the result obtained from the implementation of Gaussian Naïve Bayes Classification.

	precision	recall	f1-score	support
0	0.97	0.93	0.95	2931
1	0.21	0.41	0.28	130
accuracy			0.91	3061
macro avg	0.59	0.67	0.61	3061
weighted avg	0.94	0.91	0.92	3061

Below is the result obtained from the implementation of SVC

	precision	recall	f1-score	support
0	0.96	1.00	0.98	2931
1	0.00	0.00	0.00	130
accuracy			0.96	3061
macro avg	0.48	0.50	0.49	3061
weighted avg	0.92	0.96	0.94	3061

Below is the result obtained from the implementation of the Gradient Boost Classification.

	precision	recall	f1-score	support
0	0.96	1.00	0.98	2931
1	0.35	0.05	0.08	130
accuracy			0.96	3061
macro avg	0.66	0.52	0.53	3061
weighted avg	0.93	0.96	0.94	3061

Below is the result obtained from the implementation of the KNN.

	precision	recall	f1-score	support
0	0.96	1.00	0.98	2931
1	0.33	0.05	0.08	130
accuracy			0.96	3061
macro avg	0.65	0.52	0.53	3061
weighted avg	0.93	0.96	0.94	3061

Comparison of the models used:

Algorithm	Model Type	Data Assumption	Strengths	Weaknesses	Use Cases
Logistic Regression	Linear	Linearly separable data	Simple, interpretable, fast	Poor performance with complex patterns	Binary classification, risk prediction
Random Forest	Ensemble (Decision Trees)	No specific data assumptions	Robust to overfitting, handles non-linear	Slower, more complex, may overfit on noise	Tabular data, feature importance analysis
Gaussian Naive Bayes	Probabilistic (Bayesian)	Features are independent, Gaussian	Fast, works well with small datasets	Assumes feature independence, not flexible	Text classification, spam detection
Support Vector Classifier	Linear/Non-linear (Kernel-based)	Data may not be linearly separable	Effective in high-dimensional spaces	Memory-intensive, complex to tune	Image classification, text categorization
XGBoost	Ensemble (Gradient Boosting)	No specific assumptions	High accuracy, handles missing data	Can overfit on noisy data, complex	Structured/tabular data, competitions
Gradient Boosting	Ensemble (Boosted Trees)	No specific assumptions	Reduces bias, performs well on tabular data	Slower, sensitive to hyperparameter tuning	Credit scoring, ranking tasks

Discussion:

The XGBoost model outperformed other models due to its ability to handle class imbalance and its flexibility in modeling complex data relationships. Logistic Regression, while simple, underperformed compared to more sophisticated ensemble methods. Random Forest and SVM provided competitive results, but lacked the overall accuracy of XGBoost.

One key challenge was the imbalance in the dataset, as strokes are relatively rare events. The use of F1-score as an evaluation metric helped to account for this imbalance, ensuring that both false positives and false

negatives were minimized.

Conclusion:

This project demonstrates that machine learning models, particularly XGBoost, can effectively predict stroke risk based on patient health data. The results highlight the potential of using such models in healthcare settings to provide early warnings for stroke risk, ultimately improving patient outcomes. Future work could focus on integrating additional features, such as family medical history, to further enhance the model's predictive power.

References:

KaggleStrokePredictionDataset:
https://www.kaggle.com/competitions/playground-series-s3e2/overview

Code:

CollabLink:

https://colab.research.google.com/drive/1AjkNIsyU4WbMJbGxX-hAdylgmY4TYexv?usp=sharing

GitHub:

https://github.com/Harsha-1203/Stroke-classification