Multiclass Prediction of Obesity Risk

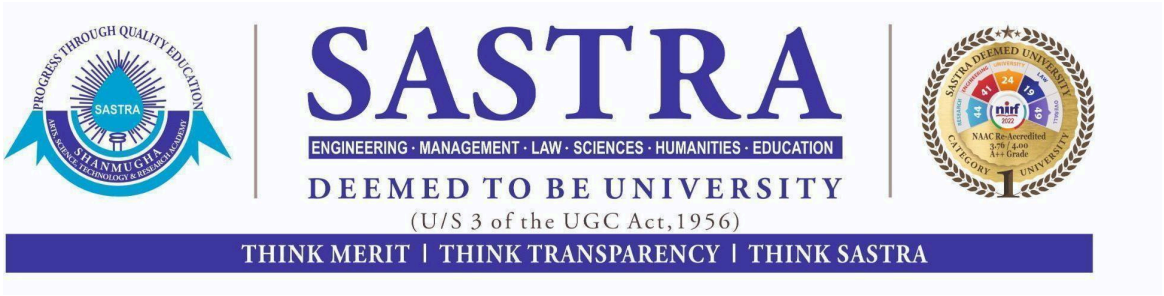
Report submitted to SASTRA Deemed to be
University As per the requirement for the course

CSE425 : MACHINE LEARNING ESSENTIALS

Submitted by

A N Harish Sai Ram
(Reg No: 125018001, B. Tech Computer Science and Business Systems)

OCTOBER- 2024



SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

Table of Contents

ABSTRACT

This project focuses on predicting obesity risk using various multiclass classification models based on health and lifestyle features such as age, weight, physical activity, and dietary habits. We implemented models like Logistic Regression, RandomForestClassifier, XGBClassifier, GradientBoostClassifier, Support Vector Classification, and Gaussian Naïve Bayes. Model performance was evaluated using accuracy, precision, recall, F1-score, and support (for weighted and macro averages). The dataset consists of over 34,000 records and 17 features with the target data being NObeyesdad.

INTRODUCTION

Importance of Dataset: As we know obesity is one of the diseases which people during middle age face, which causes a lot of restriction on diet and also leads to lifetime medication and ultimately changes their lifestyle. These days even young people are being diagnosed by it and since it's not curable, the best we can do is prevent it by predicting at the earliest to live a healthy life, which is essential in today’s day and age.

Project Objective: The objective of this project is to develop a robust machine learning model to predict obesity risk levels based on various health and lifestyle features. By leveraging advanced classification models such as Logistic Regression, RandomForestClassifier, Gaussian Naive Bayes,XGBoost and GradientBoost classification methods, the aim is to accurately classify individuals into different obesity risk categories. The project seeks to identify which models provide the highest predictive performance using evaluation metrics like accuracy, precision, recall, F1-score, and support, with a focus on improving the precision of obesity risk prediction.

Problem Statement: Predicting obesity risk is a complex task due to the multifactorial nature of obesity, which is influenced by various health and lifestyle characteristics. The problem can be framed as a multiclass classification task, where the goal is to categorize individuals into different obesity risk levels based on input features like age, weight, physical activity, and diet. Key challenges include managing multicollinearity among features, handling imbalanced classes, and choosing models that generalize well on unseen data. Additionally, missing values and categorical variables require careful preprocessing for optimal model performance.

Approach: Use classification models with a focus on predicting the obesity level of individuals by comparing the results based on 6 models.

Related Work

References:

Dataset: https://www.kaggle.com/c/playground-series-s4e2/data

WeichenLiuzz/Multi-Class-Prediction-of-Obesity-Risk

ahmedhany14/Multi-Class-Prediction-of-Obesity-Risk

julian-b24/obesity-risk-prediction

Background

- **Models**:
  - *Logistic Regression*: Logistic Regression is a classification algorithm used to predict the probability that an instance belongs to a particular class. It models the probability of a categorical dependent variable (predicting the class of sleep disorders) as a function of one or more independent variables.

  - *RandomForest Classifier:* Random Forest Classification is a method that constructs multiple decision trees during training and merges their outputs to improve predictive accuracy and control overfitting. In the code you provided, the RandomForestClassifier from scikit-learn is used to create a robust model that can handle both categorical and continuous features effectively. It aggregates the predictions from numerous decision trees to determine the final class label, leveraging the majority vote from individual trees. This approach enhances model performance, especially in complex datasets like the Sleep Health and Lifestyle Dataset, making it a powerful choice for classification tasks.

  - *Gaussian Naive Bayes*: Gaussian Naive Bayes is a probabilistic classification algorithm based on Bayes theorem, assuming that the features follow a Gaussian (normal) distribution. The GaussianNB classifier from scikit-learn is employed to predict the likelihood of various sleep disorders based on health and lifestyle features. It simplifies the computation by assuming that the presence of a particular feature is independent of the others, making it computationally efficient and suitable for

high-dimensional data. The model's performance is evaluated using metrics such as precision, recall, and F1 score, providing insights into its effectiveness in classifying sleep disorders within the dataset.

- *Support Vector Classification:* Support Vector Classification (SVC) is a supervised machine learning algorithm that seeks to find the optimal hyperplane that separates different classes in the feature space. The SVC class from scikit-learn is used to classify sleep disorders based on various lifestyle and health factors. It works well in high-dimensional spaces and is particularly effective for datasets that are not linearly separable by using kernel functions to transform the input space. The models performance is assessed using evaluation metrics like precision, recall, and F1 score, helping to determine its accuracy in predicting the correct class for sleep disorders in the dataset.

- *XGBoost Classification:* XGBoost Classification refers to the use of the XGBoost algorithm, an optimized implementation of the gradient boosting framework that is designed for speed and performance. The XGBClassifier from the XGBoost library is utilized to predict sleep disorders based on various health and lifestyle features. XGBoost improves upon traditional boosting methods by implementing regularization techniques to prevent overfitting and using parallel processing to enhance computation speed. The model's effectiveness is evaluated using metrics such as precision, recall, and F1 score, providing insights into its ability to accurately classify sleep disorders in the dataset.

- *GradientBoost Classification:* Gradient Boost Classification builds models sequentially, where each new model attempts to correct the errors made by the previous ones. The GradientBoostingClassifier from scikit-learn is used to classify sleep disorders based on various health and lifestyle factors. This method minimizes the loss function through gradient descent, making it particularly effective for handling complex relationships in the data. The performance of the model is assessed using evaluation metrics such as precision, recall, and F1 score, allowing for a comprehensive understanding of its classification accuracy in predicting sleep disorders in the dataset.

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 in this project consists of 34,598 instances.This includes a range of features that are likely useful for classification of obesity risk.

Dataset Features:

1. **id**: A unique identifier for each record.
2. **Gender**: The gender of the individual, typically 'Male' or 'Female'.
3. **Age**: The age of the individual.
4. **Height**: The height of the individual in meters.
5. **Weight**: The weight of the individual in kilograms.
6. **family_history_with_overweight**: Indicates whether the individual has a family history of being overweight (values: 'yes' or 'no').
7. **FAVC**: Frequent consumption of high-caloric food (values: 'yes' or 'no').
8. **FCVC**: Frequency of consumption of vegetables, represented by a numeric value (higher values indicate more frequent consumption).
9. **NCP**: Number of main meals per day, represented by a numeric value.
10. **CAEC**: Frequency of consuming food between meals (values: 'Always', 'Frequently', 'Sometimes', or 'Never').
11. **SMOKE**: Whether the individual smokes (values: 'yes' or 'no').
12. **CH2O**: Daily consumption of water, represented by a numeric value (in liters).
13. **SCC**: Whether the individual monitors their calorie intake (values: 'yes' or 'no').
14. **FAF**: Physical activity frequency, representing hours of physical activity per week.
15. **TUE**: Time spent using technology devices daily, represented by a numeric value (in hours).

16. **CALC**: Frequency of alcohol consumption (values: 'Always', 'Frequently', 'Sometimes', or 'Never').
17. **MTRANS**: Mode of transportation typically used (e.g., 'Walking', 'Public Transportation', Bike', 'Car').

Data Sample

id	Sex	Length	Diameter	Height	Weight	Shucked W	Viscera W	Shell Weig	Age
0	I	1.525	1.175	0.375	28.97319	12.72893	6.647958	8.348928	9
1	I	1.1	0.825	0.275	10.41844	4.521745	2.324659	3.40194	8
2	M	1.3875	1.1125	0.375	24.77746	11.3398	5.556502	6.662133	9
3	F	1.7	1.4125	0.5	50.66056	20.35494	10.99184	14.99689	11
4	I	1.25	1.0125	0.3375	23.28911	11.97766	4.507571	5.953395	8
5	M	1.5	1.175	0.4125	28.84562	13.40931	6.789705	7.93786	10
6	M	1.575	1.1375	0.35	30.02212	11.93514	7.342521	8.646598	11
7	I	1.3125	1.025	0.35	18.2996	8.249705	3.898056	5.6699	11

- **Preprocessing**:

  Preprocessing done on the data:

  Handling Missing Values: While the dataset description indicates the absence of missing values, it's

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

  Encoding Categorical Variables: Categorical features such as 'gender' and 'occupation' are

transformed into numerical representations using LabelEncoder.

Feature Scaling: Continuous variables are standardized using StandardScaler to ensure they have a

mean of zero and a standard deviation of one, which aids in model convergence and performance.

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:

Experimental Design:

- *Preprocessing*: Data cleaning, encoding, and scaling.
- *Model Training*: Split dataset into training and testing sets and train models using default parameters.
- *Model Evaluation*: Use support,F1 Score,recall and precision to assess model performance

Environment and Tools: Python, Scikit-learn, Simple Imputer, Gridsearch and Google Colab.

OPTIMIZATION DONE:

Hyperparameter Grid (`prim_grid`):

- `n_estimators`: The number of trees in the XGBoost model. GridSearch will test 100, 200, and 300 trees to find the optimal

number.

- `learning_rate`: This controls how much the model adjusts each tree's predictions. Values being tested are 0.1, 0.01, and 0.001.
- `max_depth`: The maximum depth of trees, which controls model complexity. GridSearch will test depths of 3, 5, and 7.

GridSearchCV:

- Purpose: GridSearchCV systematically works through multiple combinations of hyperparameters, cross-validating each to determine the best combination.
- Parameters:
  - `prim_grid`: The dictionary of hyperparameters to test.
  - `scoring`: The evaluation metric used for optimization (in this case, "accuracy").
  - `n_jobs=-1`: This enables parallel processing, using all available CPU cores to speed up the search.

Model Training:

- The model (`Model_XGB`) is trained on the training set (`x_train`, `y_train`) using GridSearchCV.
- After the search, the best parameters (`best_params_`) and the best score (`best_score_`) are printed.

Re-fitting the Model:

- After identifying the best hyperparameters from the grid search, a new `XGBClassifier` model is instantiated with those best parameters (i.e., `learning_rate = 0.1`, `max_depth = 5`, `n_estimators = 200`).

- The model is re-fitted on the training data (`x_train`, `y_train`).

Evaluation:

- Finally, the model's predictions on the test data (`x_test`) are compared with the true labels (`y_test`), and a classification report is generated, which includes metrics like precision, recall, and F1-score.

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: $F1\ Score = 2((Precision \cdot Recall)/(Precision + Recall))$.
- 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

	precision	recall	f1-score	support
0	0.86	0.78	0.82	580
1	0.67	0.70	0.69	605
2	0.80	0.77	0.78	564
3	0.97	0.95	0.96	668
4	1.00	0.99	1.00	807
5	0.59	0.64	0.61	447
6	0.60	0.64	0.62	481
accuracy			0.80	4152
macro avg	0.78	0.78	0.78	4152
weighted avg	0.81	0.80	0.81	4152

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

	precision	recall	f1-score	support
0	0.92	0.94	0.93	509
1	0.88	0.84	0.86	658
2	0.87	0.88	0.88	536
3	0.98	0.97	0.97	659
4	1.00	1.00	1.00	805
5	0.75	0.76	0.76	474
6	0.78	0.79	0.79	511
accuracy			0.89	4152
macro avg	0.88	0.88	0.88	4152
weighted avg	0.89	0.89	0.89	4152

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

	precision	recall	f1-score	support
0	0.85	0.70	0.77	641
1	0.47	0.65	0.54	450
2	0.61	0.37	0.46	897
3	0.93	0.71	0.81	858
4	1.00	0.96	0.98	832
5	0.30	0.61	0.40	239
6	0.24	0.53	0.33	235
accuracy			0.66	4152
macro avg	0.63	0.65	0.61	4152
weighted avg	0.74	0.66	0.68	4152

Below is the result obtained from the implementation of SVC

	precision	recall	f1-score	support
0	0.94	0.87	0.90	564
1	0.77	0.81	0.79	593
2	0.84	0.82	0.83	557
3	0.97	0.97	0.97	659
4	1.00	1.00	1.00	804
5	0.66	0.69	0.67	463
6	0.69	0.70	0.69	512
accuracy			0.85	4152
macro avg	0.84	0.83	0.84	4152
weighted avg	0.85	0.85	0.85	4152

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

	precision	recall	f1-score	support
0	0.92	0.95	0.93	506
1	0.89	0.87	0.88	639
2	0.86	0.89	0.87	527
3	0.97	0.97	0.97	656
4	1.00	0.99	1.00	807
5	0.79	0.77	0.78	494
6	0.81	0.79	0.80	523
accuracy			0.90	4152
macro avg	0.89	0.89	0.89	4152
weighted avg	0.90	0.90	0.90	4152

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

	precision	recall	f1-score	support
0	0.94	0.95	0.94	520
1	0.88	0.88	0.88	633
2	0.87	0.88	0.88	537
3	0.97	0.97	0.97	655
4	1.00	1.00	1.00	805
5	0.78	0.78	0.78	486
6	0.80	0.80	0.80	516
accuracy			0.90	4152
macro avg	0.89	0.89	0.89	4152
weighted avg	0.90	0.90	0.90	4152

Result obtained after optimization of XGBoost using GridSearch

	precision	recall	f1-score	support
0	0.92	0.95	0.93	506
1	0.89	0.87	0.88	639
2	0.86	0.89	0.87	527
3	0.97	0.97	0.97	656
4	1.00	0.99	1.00	807
5	0.79	0.77	0.78	494
6	0.81	0.79	0.80	523
accuracy			0.90	4152
macro avg	0.89	0.89	0.89	4152
weighted avg	0.90	0.90	0.90	4152

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

Result:

The XGBoost and Gradient boost classification had obtained the highest accuracies while SVC, RFC had obtained mediocre accuracies and Gaussian Naïve Bayes had obtained the lowest accuracy among them. One of the reasons could be due to its underlying assumptions that features are independent and normally distributed. If the actual feature distribution has shown any deviation from these assumptions, GNB may fail to capture the relationships within the data effectively.

Conclusion:

This project successfully explored multiple machine learning models to classify obesity risk based on health and lifestyle features. The results indicate that XGBoost and Gradient Boosting classifiers achieved the highest accuracy among the models tested, showing strong predictive performance. This suggests that ensemble methods, particularly those with gradient-boosting techniques, are well-suited for handling the complex relationships within the dataset. Models like Support Vector Classifier (SVC) and Random Forest performed moderately, while Gaussian Naïve Bayes had lower accuracy, likely due to its assumptions about feature independence and normal distribution, which may not align with the dataset's characteristics.

Links:

Colab Link:
https://colab.research.google.com/drive/1duMbLmiLcxDW52LTn9y6QhDhe0uNusPo#scrollTo=PFIbxiIjSIDU

Github Link:
https://github.com/Harish10436/ML_obesity_prediction/blob/main/multi_class_prediction_obesity_risk_6_models_90_%20(1).ipynb