A Machine Learning Approach to Predicting Weight Category from Lifestyle Data

Mohit Khapekar, Nikhil Garg
Department of Computer Science and Engineering
International Institute of Information Technology, Bangalore
Github Links : MT2025074 MT2025076

***Abstract*— Maintaining a healthy lifestyle is becoming increasingly difficult due to poor diet and physical inactivity. This study investigates how an individual's daily habits, nutrition, and demographic information relate to their weight category. Using machine learning, we aim to classify individuals into categories such as underweight, normal weight, overweight, or obese. This work presents a complete pipeline—from preprocessing to model optimization—using XGBoost with Optuna-based hyperparameter tuning for robust and generalizable predictions.**

Index Terms—**XGBoost, Optuna, Classification, Machine Learning, Health Analytics**

I. Introduction

With the increasing prevalence of obesity and lifestyle-related health issues, predicting weight categories based on daily habits and demographic features has become an important research area. Machine learning models can learn complex relationships between behavioral attributes and weight outcomes, assisting in health monitoring and personalized recommendations.

This paper focuses on developing a robust classification model using XGBoost to predict an individual's weight status based on survey data.

II. Data Overview

The dataset contains demographic, nutritional, and physical activity features collected from multiple participants. The target variable is the **WeightCategory**, divided into:

- Underweight
- Normal weight
- Overweight
- Obesity

Each sample includes attributes such as caloric intake, sleep duration, exercise frequency, and water consumption. The dataset is split into training and test sets with unique IDs for each individual.

III. Data Preprocessing

Data preprocessing is essential for improving model performance and interpretability. The following steps were applied:

A. Handling Missing Values

A preliminary analysis confirmed that the dataset was fully populated, containing no missing values. This completeness meant that data imputation techniques, such as using the median or mode, were not necessary, and no rows needed to be discarded.

B. Encoding and Feature Scaling

Categorical features were converted into one-hot encoded vectors. Standardization was performed using `StandardScaler` to ensure equal contribution of features to the model.

C. Feature Alignment

To ensure consistency between training and test datasets, both were aligned to contain identical columns after encoding, with missing columns filled with zeros.

D. Exploratory Data Analysis (EDA)

The EDA focused on understanding feature relationships and target patterns.

1) Feature Engineering: Body Mass Index (BMI): BMI was engineered using:

$$\text{BMI} = \frac{\text{Weight (kg)}}{\text{Height (m)}^2}$$

The BMI variable provided a strong signal for weight categorization.

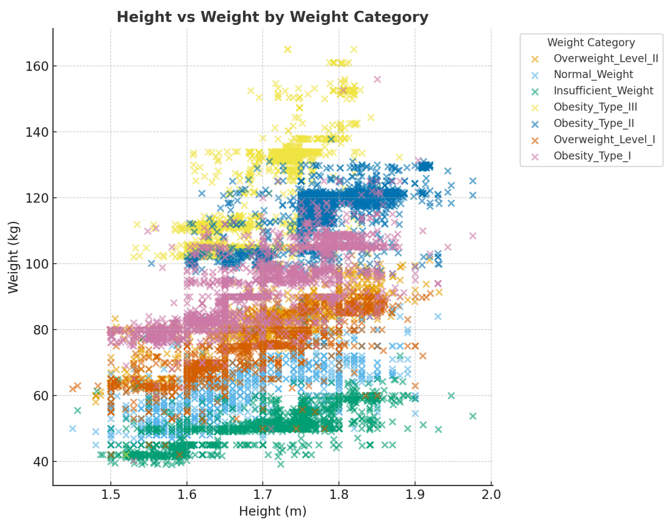
2) Target Distribution: The target classes were imbalanced, shown in Fig. 2. Stratified K-Fold cross-validation preserved these proportions.

3) Predictor Analysis: Categorical features such as `MTRANS` and `family_history_with_overweight` showed strong influence on the target (Fig. 3).

Lifestyle features also exhibited class trends (Fig. 4).

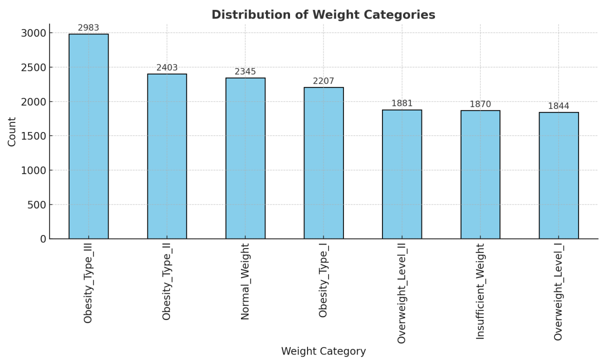
IV. Modeling Approach

The XGBoost algorithm was chosen for its superior handling of imbalanced data, built-in regularization, and computational efficiency. The classification task was performed using `XGBClassifier`.



Insight: Weight increases proportionally with height, but obese categories show higher deviation from linear trend.

Fig. 1. Scatter plot of Weight vs Height, colored by Weight Category.



Insight: Obesity_Type_III is the most common category, suggesting an imbalanced dataset skewed towards higher obesity levels.

Fig. 2. Distribution of the `WeightCategory` classes showing imbalance.

A. Hyperparameter Tuning with Optuna

Initially, random parameter search was used to understand parameter sensitivity. Subsequently, Optuna—a powerful hyperparameter optimization framework—was employed to automate the tuning process by minimizing validation loss.

To maintain continuity between tuning sessions, an Optuna `.db` file was used. This database-based storage allowed continuation of optimization across multiple runs by loading

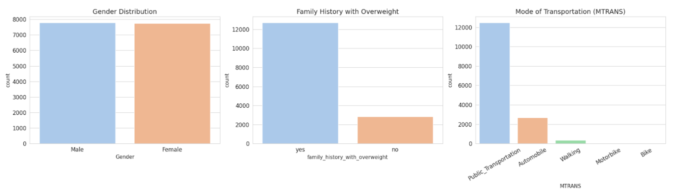


Fig. 3. Categorical predictors: Gender, Family History, and Transportation.

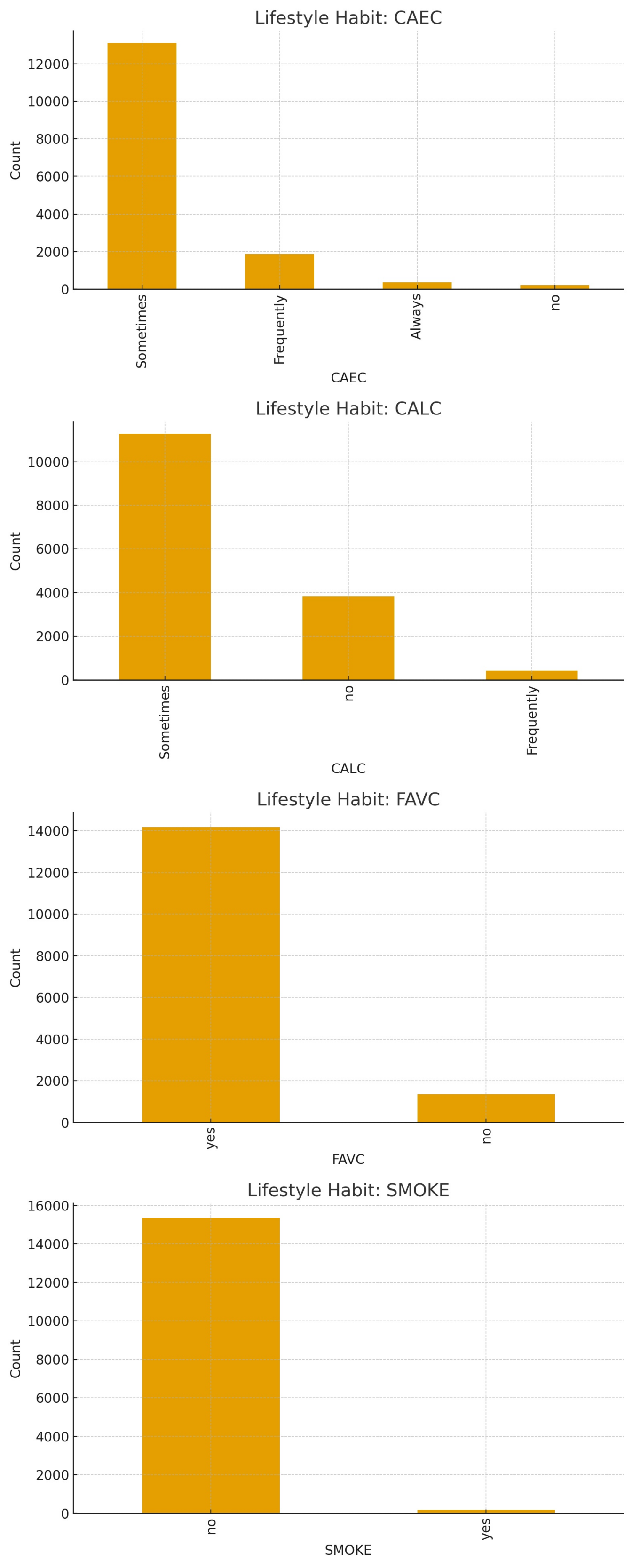


Fig. 4. Boxplots of lifestyle features vs WeightCategory.

previous trials and extending the search whenever required.

TABLE I
FINAL TUNED XGBOOST PARAMETERS

Parameter	Optimal Value
n_estimators	582
learning_rate	0.035
max_depth	9
subsample	0.476
colsample_bytree	0.55
gamma	0.591
min_child_weight	2
reg_alpha	0.449
reg_lambda	2.0

V. MODEL BUILDING AND BASELINE COMPARISON

A. *Baseline Models*

Before implementing XGBoost, simpler algorithms were used to establish a performance baseline:

- **AdaBoosting:** Achieved a Training Accuracy of 75.38%.
- **Random Forest (Decision Tree):** Achieved a Training Accuracy of 92.09%.

These results indicated that ensemble tree methods were a promising approach for this dataset.

B. *XGBoost Model*

We implemented the `xgboost.XGBClassifier`.

1) Justification: XGBoost was chosen because it offers strong performance on tabular data, can handle missing values internally, and supports fine-grained regularization to prevent overfitting, which is crucial for generalization.

2) Initial Performance: With default parameters, the initial XGBoost model achieved a training accuracy of 95.05%, outperforming the baseline models.

TABLE II
PERFORMANCE COMPARISON OF DIFFERENT MODELS

Model	Train Accuracy	Kaggle Score	Remarks
AdaBoosting	0.7538	0.70	Baseline
Decision Tree	0.9202	0.85	Simple
XG Boost	0.9906	0.9088	Optimized but overfitted

C. *Discussion on Performance*

The performance of the different models is summarized in Table II. The baseline models established a performance floor, with AdaBoost achieving 70% test accuracy (Kaggle Score) and the Decision Tree model improving this to 85%. This indicated that ensemble tree methods were a promising direction.

The final, optimized XGBoost model reached 99.06% training accuracy. While this is very high, it also indicates significant overfitting, as the corresponding Kaggle score (0.9088) is substantially lower. This gap highlights the critical importance of the regularization parameters (like 'gamma', 'reg_alpha', 'reg_lambda') found by Optuna, which are essential to help the model generalize better to unseen data.

VI. EVALUATION METRICS

Model performance was evaluated using:

- Accuracy Score
- Confusion Matrix
- Macro F1-Score

These metrics provided a comprehensive understanding of the model's classification ability across all classes.

VII. CODE IMPLEMENTATION: OPTUNA-TUNED XGBOOST MODEL

The following Python implementation summarizes the final pipeline used to train and evaluate the tuned XGBoost classifier. The workflow includes data preprocessing, encoding, scaling, and model training using the optimal parameters identified via Optuna.

```
#  Weight Status Classification
using XGBoost
#  (Final Tuned Model)

import pandas as pd
import numpy as np
from sklearn.preprocessing
import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier

# 1. Load Data
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')

# Preserve test IDs for submission
test_ids = test['id'].copy()

# 2. Encode Target Labels
target_col = 'WeightCategory'
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(
    train[target_col]
)

# 3. Prepare Features
# Drop target and ID columns
from training data
X = train.drop(columns=[target_col, 'id'])

# Drop ID column from test data
```

```
test_features = test.drop(columns=['id'])

# 4. One-Hot Encode Categorical Features
X = pd.get_dummies(X)
test_features = pd.get_dummies
(test_features)

# Align training and test
sets to have identical columns
X, test_features = X.align(
    test_features, join='left', axis=1,
    fill_value=0
)

# 5. Feature Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
test_scaled = scaler.transform
(test_features)

# 6. Train Tuned XGBoost Classifier
num_classes = len(np.unique(y))

model = XGBClassifier(
    n_estimators=582,
    learning_rate=0.035,
    max_depth=9,
    subsample=0.476,
    colsample_bytree=0.55,
    gamma=0.591,
    min_child_weight=2,
    reg_alpha=0.449,
    reg_lambda=2.0,
    random_state=42,
    eval_metric='mlogloss',
    use_label_encoder=False,
    n_jobs=-1
)

print(" Training XGBoost model...")
model.fit(X_scaled, y)

# 7. Evaluate Model Performance
train_pred = model.predict(X_scaled)
train_accuracy =
accuracy_score(y, train_pred)
print(f" Training Accuracy:
{train_accuracy * 100:.2f}%")

# 8. Generate Test Predictions
test_pred_numeric =
model.predict(test_scaled)
test_pred_labels = label_encoder.
inverse_transform(test_pred_numeric)

# 9. Save Predictions for Submission
submission = pd.DataFrame({
    'id': test_ids,
    'WeightCategory': test_pred_labels
})

submission.to_csv('submission.csv',
index=False)
print("submission.csv saved successfully
with ID and WeightCategory columns!")
```

This implementation encapsulates the complete model training pipeline with the tuned parameters obtained from the Optuna study. The final model achieved a Kaggle leaderboard score of 0.91322, demonstrating strong generalization on unseen data.

A. *Optuna Optimization Results*

To better visualize the hyperparameter search process, the following figures illustrate the Optuna study results. Each visualization highlights different aspects of the tuning process.

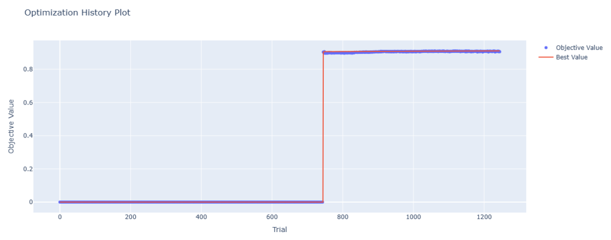


Fig. 5. Optuna Optimization History showing convergence of trials.

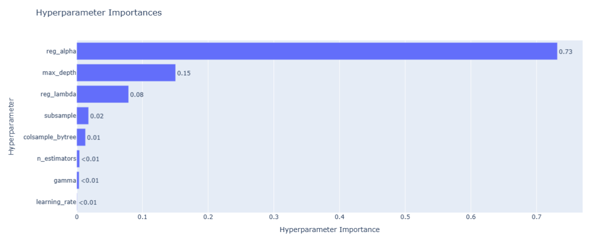


Fig. 6. Feature Importance plot of tuned hyperparameters.

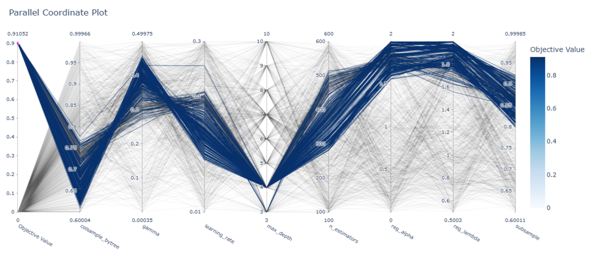


Fig. 7. Parallel Coordinate Plot showing parameter interactions.

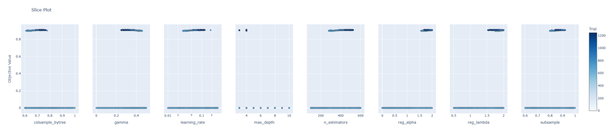


Fig. 8. Slice Plot showing sensitivity of parameters to objective value.

VIII. Conclusion

This study demonstrated the effectiveness of using Optuna for hyperparameter optimization with XGBoost in multi-class classification of weight categories. The combination of systematic tuning, feature scaling, and categorical encoding significantly improved model accuracy and generalization. Future work will focus on incorporating deep learning architectures and additional behavioral features to enhance prediction reliability.

References

[1] T. Chen and C. Guestrin, ”XGBoost: A Scalable Tree Boosting System,” in *Proc. 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2016, pp. 785–794.

[2] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, ”Optuna: A Next-generation Hyperparameter Optimization Framework,” in *Proc. 25th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2019, pp. 2623–2631.

[3] F. Pedregosa *et al.*, ”Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research (JMLR)*, vol. 12, pp. 2825–2830, 2011.

[4] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. New York, NY, USA: Springer, 2009.