Obesity in South America

Estimation through eating habits and physical conditions

530463957 mjia0724

520497760 czho0687

541029720 spen0159

540823284 nlee0431

Our study examines the predictive relationship between lifestyle factors (such as eating habits and physical activity) and obesity in South America. This analysis leverages machine learning to understand how well obesity levels can be inferred from variables in a dataset representing 2111 entries from Mexico, Peru, and Colombia. The dataset includes both original survey data and additional synthetic data generated through SMOTE to balance classes, ensuring that underrepresented weight categories, such as severe obesity, are adequately represented.

Our target variable, obesity level (in the "NObeyesdad" column), is categorized by BMI-based classifications. While BMI serves as a primary health indicator, we refined our analysis by selecting only certain demographic and lifestyle attributes, excluding height and weight. In creating our models, the data was partitioned into training and test sets, with a fixed 10% for testing and a random seed set for consistent, replicable outcomes.

Two evaluation metrics, confusion matrix and top-2 accuracy, were chosen for their ability to provide nuanced insight into classification accuracy, especially in multiclass settings. This selection aligns with our primary focus on exploring obesity categorization rather than focusing on fine-grained BMI distinctions. The exploratory nature of this report aims to shed light on the underlying lifestyle correlations without asserting causality, offering a foundation for future health policy considerations.

SID – 541029720 UNIKEY – SPEN0159

Data Set Overview

This dataset [1] has been acquired from the website UC Irvine Machine Learning Repository. This dataset include data for the estimation of obesity levels in individuals from the countries of Mexico, Peru and Colombia, based on their eating habits and physical condition. The data contains 17 attributes and 2111 records, the records are labelled with the class variable NObesity (Obesity Level-target variable), that allows classification of the data using the values of Insufficient Weight, Normal Weight, Overweight Level I, Overweight Level II, Obesity Type I, Obesity Type II and Obesity Type III. 77% of the data was generated synthetically using the Weka tool and the SMOTE filter, 23% of the data was collected directly from users through a web platform. The data set was initially accessed on the 20th October, 2024. This dataset is licensed under a Creative Commons Attribution 4.0 International (CC BY 4.0) license.

Data Dictionary

Gender (Categorical) – Classification of gender
Age (Continuous) - Ranging from 14 – 61 (inclusive of 14 and 61).
Height (Continuous) – Ranging from 145 cm – 198 cm
Weight (Continuous) – Ranging from 39kgs – 173kgs
Family_history_with_overweight (Binary) - Has a family member suffered or suffers from overweight?
FAVC (Binary) - Do you eat high caloric food frequently?
FCVC (Integer) - Do you usually eat vegetables in your meals?
NCP (Continuous) - How many main meals do you have daily?
CAEC (Categorical) - Do you eat any food between meals?
SMOKE (Binary) - Do you smoke?
CH2O (Continuous) - How much water do you drink daily ?
SCC (Binary) - Do you monitor the calories you eat daily?
FAF (Continuous) - How often do you have physical activity?
TUE (Integer) - How much time do you use technological devices such as cell phone, videogames, television, computer and others?
CALC (Categorical) - How often do you drink alcohol?
MTRANS (Categorical) - Which transportation do you usually use?
NObeyesdad (Categorical) - Obesity level

The data has been split into a test size of 0.1 which means 10 percent of the data was used for testing and 90 percent of the data was used for training. X consists of all the input data (every variable except NObeyesdad. y contains the target variable, the variable we are trying to predict. The random state has been set to 42 allowing for the data split to be reproducible.

Machine Learning Models

To predict the NObeyesdad variable, I used **neural networks** and **a support vector machine.**

[1] UCI Machine Learning Repository. (2019, August 26). Retrieved from archive.ics.uci.edu website: https://archive.ics.uci.edu/dataset/544/estimation+of+obesity+levels+based+on+eating+habits+and+physical+condition

#import the necessary libraries

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from collections import Counter
from sklearn.metrics import f1_score, confusion_matrix
from sklearn.svm import SVC
```

#define the file path for the csv file

```
file_path = '/Users/shrikha/Desktop/project3/ObesityDataSet_raw_and_data_sinthetic.csv'
```

#read the file path

```
df = pd.read_csv(file_path)
```

#remove the column for weight and height from the data frame (BMI is derived from these two attributes (BMI = Weight / (Height^2)), keeping both may lead ***to multicollinearity****.)*

```
df.drop('Height', axis=1, inplace=True)
df.drop('Weight', axis=1, inplace=True)
```

```
from sklearn import preprocessing
```

#initialise a dictionary to hold the encoding for the categorical columns

```
label_encoders = {}
categorical_columns = ['NObeyesdad', 'Gender', 'family_history_with_overweight', 'FAVC',
'CAEC', 'SMOKE', 'SCC', 'CALC', 'MTRANS']
```

#loop through the columns, create a label encoder and transform the columns

```
for column in categorical_columns:
    label_encoders[column] = preprocessing.LabelEncoder()
    df[column] = label_encoders[column].fit_transform(df[column])
```

#define the feature and target variables

```
X = df.drop(columns=['NObeyesdad'])
y = df['NObeyesdad']
```

#split the dataset into training and testing set

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)
print(f"Training set shape: {X_train.shape}, {y_train.shape}")
print(f"Testing set shape: {X_test.shape}, {y_test.shape}")
```

```
from sklearn.preprocessing import StandardScaler
```

#standardise the feature data

```
scaler = StandardScaler()
scaler.fit(X_train)
```

#apply the scaler to the data

```
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

#initialise and train the MLPClassifier

```
from sklearn.neural_network import MLPClassifier
model = MLPClassifier(hidden_layer_sizes =(10,10,10), max_iter = 1000)
model.fit(X_train, y_train.values.ravel())
```

#make predictions with the model

```
predictions = model.predict(X_test)
```

#print the predictions

```
print(predictions)
```

#decode the encoded labels

```
decoded_predictions = label_encoders['NObeyesdad'].inverse_transform(predictions)

print("Decoded Predictions:", decoded_predictions.tolist())
```

#count the number of time each label occurs

```
prediction_counts = Counter(decoded_predictions)
print("Count of Predictions:", prediction_counts)
```

```
Training set shape: (1899, 14), (1899,)
Testing set shape: (212, 14), (212,)
[0 4 0 0 6 0 3 5 2 0 2 4 3 3 4 1 4 3 3 2 2 5 4 2 1 6 6 4 4 0 4 5 5 6 0 4 2
 1 1 1 2 3 6 1 5 0 3 2 0 3 1 2 1 5 0 1 3 3 1 3 1 4 3 0 5 1 1 4 0 1 6 6 0 0
 2 6 3 4 4 3 1 6 3 3 2 6 2 4 6 1 1 2 0 1 6 0 0 2 1 3 5 2 4 2 6 2 0 4 4 3 6
 5 2 2 0 2 2 3 6 5 4 3 2 6 3 3 2 2 4 3 4 3 1 2 5 4 5 4 1 2 5 5 0 6 5 3 2 4
 0 2 0 0 0 3 2 0 6 2 0 0 1 1 2 0 3 0 2 1 5 6 0 1 3 4 2 6 2 6 2 0 0 3 0 5 3
 2 5 4 6 3 0 4 0 1 4 1 3 2 4 0 3 3 4 4 6 4 5 5 2 4 5 2]
Decoded Predictions: ['Insufficient_Weight', 'Obesity_Type_III', 'Insufficient_Weight', 'Insufficient_Weight', 'Overweight_Level_II', 'Insufficient_Weight', 'Obesity_Type_II', 'Overweight_Level_I', 'Obesity_Type_I',
 'Insufficient_Weight', 'Obesity_Type_I', 'Obesity_Type_III', 'Obesity_Type_II', 'Obesity_Type_II', 'Obesity_Type_III', 'Normal_Weight', 'Obesity_Type_III', 'Obesity_Type_II', 'Obesity_Type_II', 'Obesity_Type_I', 'O
besity_Type_I', 'Overweight_Level_I', 'Obesity_Type_III', 'Obesity_Type_I', 'Normal_Weight', 'Overweight_Level_II', 'Overweight_Level_II', 'Obesity_Type_III', 'Obesity_Type_III', 'Insufficient_Weight', 'Obesity_Type
_III', 'Overweight_Level_I', 'Overweight_Level_I', 'Overweight_Level_II', 'Insufficient_Weight', 'Obesity_Type_III', 'Obesity_Type_I', 'Normal_Weight', 'Normal_Weight', 'Obesity_Type_I', 'Obesity_Ty
pe_II', 'Overweight_Level_II', 'Normal_Weight', 'Overweight_Level_I', 'Insufficient_Weight', 'Obesity_Type_II', 'Obesity_Type_I', 'Insufficient_Weight', 'Obesity_Type_II', 'Normal_Weight', 'Obesity_Type_I', 'Normal_
Weight', 'Overweight_Level_I', 'Insufficient_Weight', 'Normal_Weight', 'Obesity_Type_II', 'Obesity_Type_II', 'Normal_Weight', 'Obesity_Type_II', 'Normal_Weight', 'Obesity_Type_III', 'Obesity_Type_II', 'Insufficient_
Weight', 'Overweight_Level_I', 'Normal_Weight', 'Normal_Weight', 'Obesity_Type_III', 'Insufficient_Weight', 'Normal_Weight', 'Overweight_Level_II', 'Overweight_Level_II', 'Insufficient_Weight', 'Insufficient_Weight'
, 'Obesity_Type_I', 'Overweight_Level_II', 'Obesity_Type_II', 'Obesity_Type_III', 'Obesity_Type_III', 'Obesity_Type_II', 'Normal_Weight', 'Overweight_Level_II', 'Obesity_Type_II', 'Obesity_Type_II', 'Obesity_Type_I'
, 'Overweight_Level_II', 'Obesity_Type_I', 'Obesity_Type_III', 'Overweight_Level_II', 'Normal_Weight', 'Normal_Weight', 'Obesity_Type_II', 'Insufficient_Weight', 'Normal_Weight', 'Overweight_Level_II', 'Insufficient
_Weight', 'Insufficient_Weight', 'Obesity_Type_I', 'Normal_Weight', 'Obesity_Type_II', 'Overweight_Level_I', 'Obesity_Type_I', 'Obesity_Type_III', 'Obesity_Type_I', 'Overweight_Level_II', 'Obesity_Type_I', 'Insuffic
ient_Weight', 'Obesity_Type_III', 'Obesity_Type_III', 'Obesity_Type_II', 'Overweight_Level_II', 'Overweight_Level_II', 'Obesity_Type_I', 'Obesity_Type_I', 'Insufficient_Weight', 'Obesity_Type_I', 'Obesity_Type_I', 'Q
besity_Type_II', 'Overweight_Level_II', 'Overweight_Level_I', 'Obesity_Type_III', 'Obesity_Type_II', 'Obesity_Type_I', 'Overweight_Level_II', 'Obesity_Type_II', 'Obesity_Type_II', 'Obesity_Type_I', 'Obesity_Type_I',
 'Obesity_Type_III', 'Obesity_Type_II', 'Obesity_Type_III', 'Obesity_Type_II', 'Normal_Weight', 'Obesity_Type_I', 'Overweight_Level_I', 'Obesity_Type_III', 'Overweight_Level_I', 'Obesity_Type_III', 'Normal_Weight',
 'Obesity_Type_I', 'Overweight_Level_I', 'Overweight_Level_I', 'Insufficient_Weight', 'Overweight_Level_II', 'Overweight_Level_I', 'Obesity_Type_II', 'Obesity_Type_I', 'Obesity_Type_III', 'Insufficient_Weight', 'Obes
ity_Type_I', 'Insufficient_Weight', 'Insufficient_Weight', 'Insufficient_Weight', 'Obesity_Type_II', 'Obesity_Type_I', 'Insufficient_Weight', 'Overweight_Level_II', 'Obesity_Type_I', 'Insufficient_Weight', 'Insuffic
ient_Weight', 'Normal_Weight', 'Normal_Weight', 'Obesity_Type_I', 'Insufficient_Weight', 'Obesity_Type_II', 'Insufficient_Weight', 'Obesity_Type_I', 'Normal_Weight', 'Overweight_Level_I', 'Overweight_Level_II', 'Ins
ufficient_Weight', 'Normal_Weight', 'Obesity_Type_II', 'Obesity_Type_III', 'Obesity_Type_I', 'Overweight_Level_II', 'Obesity_Type_I', 'Overweight_Level_II', 'Obesity_Type_I', 'Insufficient_Weight', 'Insufficient_Wei
ght', 'Obesity_Type_II', 'Insufficient_Weight', 'Overweight_Level_I', 'Obesity_Type_II', 'Obesity_Type_I', 'Overweight_Level_I', 'Obesity_Type_III', 'Overweight_Level_II', 'Obesity_Type_II', 'Insufficient_Weight', '
Obesity_Type_III', 'Insufficient_Weight', 'Normal_Weight', 'Obesity_Type_III', 'Normal_Weight', 'Obesity_Type_II', 'Obesity_Type_I', 'Obesity_Type_III', 'Insufficient_Weight', 'Obesity_Type_II', 'Obesity_Type_II', '
Obesity_Type_III', 'Obesity_Type_III', 'Overweight_Level_II', 'Obesity_Type_III', 'Overweight_Level_I', 'Overweight_Level_I', 'Obesity_Type_I', 'Obesity_Type_III', 'Overweight_Level_I', 'Obesity_Type_I']
Count of Predictions: Counter({'Obesity_Type_I': 38, 'Insufficient_Weight': 36, 'Obesity_Type_II': 35, 'Obesity_Type_III': 32, 'Normal_Weight': 27, 'Overweight_Level_II': 23, 'Overweight_Level_I': 21})
```

#print the results

```
print("MLP Confusion Matrix:\n", confusion_matrix(y_test, predictions))
mlp_probabilities = model.predict_proba(X_test)
# Calculate the top-k accuracy score using the predicted probabilities
accuracy = top_k_accuracy_score(y_test, mlp_probabilities, k=2)
print("MLP Top K accuracy Score:", accuracy)
mlp_accurary_score = accuracy_score(y_test, predictions)
print(f"Accuracy score:{mlp_accurary_score}")
```

```
MLP Confusion Matrix:
 [[29  0  0  0  0  1  1]
 [ 3 17  5  1  1  3  5]
 [ 1  1 30  1  0  1  4]
 [ 0  0  0 32  0  1  2]
 [ 0  0  0  0 30  0  0]
 [ 3  5  2  0  0 12  0]
 [ 0  4  1  1  1  3 11]]
MLP Top K accuracy Score: 0.8773584905660378
```

#create a linear svm model

```
svm_model = SVC(kernel='linear')
```

#train the data

```
svm_model.fit(X_train, y_train)
```

#make predictions with the model

```
svm_predictions = svm_model.predict(X_test)

print(svm_predictions)
```

#decode the encoded labels

```
decoded_svm_predictions = label_encoders['NObeyesda'].inverse_transform(svm_predictions)

print("SVM Decoded Predictions:", decoded_svm_predictions.tolist())
```

#count the number of time each label occurs

```
svm_prediction_counts = Counter(decoded_svm_predictions)
```

```
SVM Decoded Predictions: ['Insufficient_Weight', 'Obesity_Type_III', 'Overweight_Level_I', 'Insufficient_Weight', 'Overweight_Level_II', 'Overweight_Level_I', 'Obesity_Type_II',
 'Obesity_Type_I', 'Obesity_Type_I', 'Insufficient_Weight', 'Obesity_Type_I', 'Obesity_Type_III', 'Obesity_Type_II', 'Obesity_Type_II', 'Obesity_Type_III', 'Insufficient_Weight',
 'Obesity_Type_III', 'Obesity_Type_II', 'Obesity_Type_II', 'Obesity_Type_I', 'Obesity_Type_I', 'Obesity_Type_I', 'Obesity_Type_III', 'Obesity_Type_I', 'Insufficient_Weight', 'Over
weight_Level_II', 'Obesity_Type_I', 'Obesity_Type_III', 'Obesity_Type_III', 'Insufficient_Weight', 'Obesity_Type_III', 'Overweight_Level_II', 'Overweight_Level_I', 'Overweight_Le
vel_II', 'Insufficient_Weight', 'Obesity_Type_III', 'Obesity_Type_I', 'Normal_Weight', 'Insufficient_Weight', 'Normal_Weight', 'Obesity_Type_I', 'Obesity_Type_II', 'Overweight_Le
vel_II', 'Normal_Weight', 'Obesity_Type_I', 'Insufficient_Weight', 'Obesity_Type_II', 'Obesity_Type_I', 'Overweight_Level_I', 'Obesity_Type_II', 'Normal_Weight', 'Normal_Weight',
 'Normal_Weight', 'Overweight_Level_I', 'Insufficient_Weight', 'Overweight_Level_II', 'Obesity_Type_II', 'Obesity_Type_II', 'Overweight_Level_I', 'Obesity_Type_II', 'Insufficient
_Weight', 'Obesity_Type_III', 'Obesity_Type_II', 'Insufficient_Weight', 'Obesity_Type_I', 'Normal_Weight', 'Normal_Weight', 'Obesity_Type_III', 'Insufficient_Weight', 'Insufficie
nt_Weight', 'Obesity_Type_I', 'Obesity_Type_I', 'Insufficient_Weight', 'Insufficient_Weight', 'Obesity_Type_I', 'Obesity_Type_II', 'Obesity_Type_II', 'Obesity_Type_I', 'Obesity_T
ype_III', 'Obesity_Type_II', 'Normal_Weight', 'Obesity_Type_I', 'Obesity_Type_II', 'Obesity_Type_II', 'Obesity_Type_I', 'Obesity_Type_I', 'Obesity_Type_I', 'Obesity_Type_I', 'Ove
rweight_Level_II', 'Overweight_Level_II', 'Normal_Weight', 'Obesity_Type_II', 'Normal_Weight', 'Normal_Weight', 'Insufficient_Weight', 'Insufficient_Weight', 'Insufficient_Weight
', 'Obesity_Type_I', 'Normal_Weight', 'Obesity_Type_II', 'Obesity_Type_II', 'Obesity_Type_I', 'Obesity_Type_I', 'Obesity_Type_III', 'Obesity_Type_I', 'Overweight_Level_II', 'Obesity_Type_I', 'Insu
fficient_Weight', 'Obesity_Type_III', 'Obesity_Type_III', 'Obesity_Type_II', 'Obesity_Type_I', 'Normal_Weight', 'Obesity_Type_I', 'Overweight_Level_I', 'Insufficient_Weight', 'Ob
esity_Type_I', 'Obesity_Type_I', 'Obesity_Type_II', 'Overweight_Level_II', 'Obesity_Type_I', 'Obesity_Type_I', 'Obesity_Type_III', 'Obesity_Type_I', 'Overweight_Level_I', 'Overw
eight_Level_II', 'Obesity_Type_II', 'Obesity_Type_I', 'Obesity_Type_II', 'Obesity_Type_III', 'Obesity_Type_II', 'Obesity_Type_III', 'Obesity_Type_II', 'Normal_Weight', 'Obesity_T
ype_I', 'Overweight_Level_I', 'Obesity_Type_III', 'Obesity_Type_I', 'Obesity_Type_III', 'Insufficient_Weight', 'Obesity_Type_I', 'Obesity_Type_II', 'Overweight_Level_I', 'Insuffi
cient_Weight', 'Insufficient_Weight', 'Overweight_Level_II', 'Obesity_Type_II', 'Obesity_Type_I', 'Obesity_Type_III', 'Insufficient_Weight', 'Obesity_Type_I', 'Insufficient_Weigh
t', 'Normal_Weight', 'Insufficient_Weight', 'Obesity_Type_II', 'Obesity_Type_I', 'Obesity_Type_I', 'Normal_Weight', 'Obesity_Type_II', 'Insufficient_Weight', 'Insufficient_Weight
', 'Insufficient_Weight', 'Normal_Weight', 'Obesity_Type_I', 'Insufficient_Weight', 'Obesity_Type_II', 'Insufficient_Weight', 'Obesity_Type_II', 'Normal_Weight', 'Overweight_Leve
l_I', 'Overweight_Level_II', 'Obesity_Type_I', 'Normal_Weight', 'Obesity_Type_II', 'Obesity_Type_III', 'Obesity_Type_I', 'Overweight_Level_II', 'Obesity_Type_I', 'Obesity_Type_I'
, 'Obesity_Type_II', 'Normal_Weight', 'Insufficient_Weight', 'Obesity_Type_II', 'Insufficient_Weight', 'Overweight_Level_I', 'Obesity_Type_II', 'Obesity_Type_II', 'Overweight_Lev
el_I', 'Obesity_Type_III', 'Overweight_Level_II', 'Obesity_Type_II', 'Insufficient_Weight', 'Obesity_Type_III', 'Obesity_Type_I', 'Insufficient_Weight', 'Obesity_Type_III', 'Norm
al_Weight', 'Obesity_Type_II', 'Overweight_Level_I', 'Obesity_Type_III', 'Overweight_Level_I', 'Obesity_Type_II', 'Obesity_Type_II', 'Obesity_Type_III', 'Obesity_Type_III', 'Norm
al_Weight', 'Obesity_Type_III', 'Overweight_Level_I', 'Normal_Weight', 'Obesity_Type_I', 'Obesity_Type_III', 'Normal_Weight', 'Obesity_Type_I']
SVM Count of Predictions: Counter({'Obesity_Type_I': 48, 'Obesity_Type_II': 42, 'Insufficient_Weight': 36, 'Obesity_Type_III': 30, 'Normal_Weight': 25, 'Overweight_Level_I': 16,
'Overweight_Level_II': 15})
```

```
print("SVM Count of Predictions:", svm_prediction_counts)
```

#use top k accuracy score and confusion matrix to evaluate the model's performance
#print the results

```
print("SVM Confusion Matrix:\n", confusion_matrix(y_test, svm_predictions))
svm_probabilities = svm_model.predict_proba(X_test)
svm_accuracy = top_k_accuracy_score(y_test, svm_probabilities, k=2)
svm_accuracy_score = accuracy_score(y_test, svm_predictions)
print("SVM Top K accuracy Score:", svm_accuracy)
print(f"Accuracy score:{svm_accuracy_score}")
```

```
SVM Confusion Matrix:
 [[24  2  2  0  0  3  0]
 [ 8 14  6  0  0  1  6]
 [ 1  1 28  6  0  2  0]
 [ 0  0  1 33  0  0  1]
 [ 0  0  0  0 30  0  0]
 [ 2  6  5  2  0  6  1]
 [ 1  2  6  1  0  4  7]]
SVM Top K accuracy Score: 0.8160377358490566
```

Analysis

As this is a multiclass classification case, the models used must be able to deal with complex patterns in the data. Classification MLPs are versatile and powerful, capable of distinguishing between multiple classes, making them invaluable for a wide range of applications, from digit recognition to complex object classification. They capture the non-linear relationship between the variables. The hidden layers allow the model to train and learn intricate patterns. Support Vector Machine Model is a powerful and widely used machine learning algorithm that can be used for classification, regression, and outlier detection. It is extremely good at finding the optimal decision boundary between different classes by maximizing the margin between them. SVM works by finding a hyperplane in a high-dimensional space that maximally separates different classes. This hyperplane can then be used to classify new data points based on their position relative to the hyperplane. Both the MLP and SVM models can be further improved by tuning hyperparameters. The hidden layers in MLP can be increased and learning rates can be adjusted for optimal performance. For SVM, changing kernal functions or adjusting the regularization function can lead to improved performances.

Evaluation of models

For the MLP model the confusion matrix has 29 correct predictions in class 1, class 2 has 17 correct predictions, class 3 has 30 correct predictions etc (along the leading diagonal). The off-diagonal elements represent misclassifications. In this case there are 46 misclassifications. The overall performance according to the accuracy score is that the model has 71 percent accuracy (correct classification of labels) and the k top accuracy score suggests that 87 percent of the time, the true class was in top 2 predictions.

For the SVM model the confusion matrix has 24 correct predictions in class 1, class 2 has 14 correct predictions, class 3 has 18 correct predictions etc (along the leading diagonal). The off-diagonal elements represent misclassifications. In this case there are 71 misclassifications. The overall performance according to the accuracy score is that the model has 68 percent

accuracy (correct classification of labels) and the k top accuracy score suggests that 81 percent of the time, the true class was in top 2 predictions.
The MLP model displays more accurate values than the SVM model. This can be attributed to different reasons such as scalability of large datasets where MLP models perform better with increase in training and SVM models' performance can decrease with larger datasets. MLPs are more flexible when it comes to dealing with multiclass classification. While, SVMs are more flexible with binary classification.

1. Introduction

This analysis seeks to predict the levels of obesity using a dataset from the UCI Machine Learning Repository titled *Estimation of Obesity Levels Based on Eating Habits and Physical Condition* [1]. The dataset contains 17 attributes, including demographic information, eating habits, and physical activity levels, with the target variable being *NObeyesdad*, categorized into multiple classes for obesity levels.

To avoid data leakage, as the classification of obesity is directly provided by using BMI, *Height* and *Weight* were excluded from processing the data. The dataset was split into a training set containing 90% of the data and a test set with the remaining 10%. Model stability was evaluated by cross-validation. Overall accuracy, Top-2 accuracy, and confusion matrices are the main metrics considered, given that they provide insights with respect to the performance of the models in effectively distinguishing different levels of obesity.

Preprocessing

The data was initially loaded into a Pandas DataFrame. I then removed the columns 'Weight' and 'Height' because those are used in calculating the predictor variable for levels of obesity. To handle categorical features, label encoding was done so it would turn them into numerical data and be model-friendly. Also, I encoded the target variable 'NObeyesdad' as discrete labels. The column 'Age' has been normalized using 'StandardScaler' to make sure that all the features contribute equally in analysis. Finally, it splitting into training and testing sets was necessary for efficiency of performance testing of predictive models.

```
data = data.drop(['Weight', 'Height'], axis=1)
for column in categorical_columns:
    label[column] = LabelEncoder()
    data.loc[:, column] = label[column].fit_transform(data[column])
label['NObeyesdad'] = LabelEncoder()
data['NObeyesdad'] = label['NObeyesdad'].fit_transform(data['NObeyesdad'])
scaler = StandardScaler()
data['Age'] = scaler.fit_transform(data[['Age']])
X = data.drop('NObeyesdad', axis=1)
y = data['NObeyesdad']  # encoded as integers
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)
```

2. Model Selection

For this analysis, I will use Logistic Regression and the Bagging Classifier to predict the level of obesity based on health and lifestyle attributes. Here, I have chosen Logistic Regression for its interpretability, speed, and it being a good baseline model for multi-class classification, one-vs-rest, to determine which level of obesity is being compared. It is also linear in nature; thus, giving direct insight into how each feature influences the model outcome is well-suited for preliminary analysis. While the Bagging Classifier uses ensemble learning to improve predictive accuracy by reducing overfitting and variance, it effectively models complex nonlinear relationships inherent in the dataset. This combination of models balances simplicity and robustness to provide comprehensive insight into obesity prediction in a multi-class setting.

3. Model Approaches and Evaluation

2.1 Logistic Regression

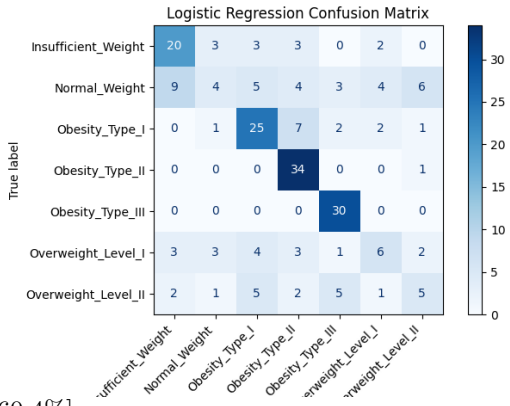
Logistic Regression is a linear classifier that can be used to estimate the probability of a sample belonging to a certain class based on a set of independent variables. For this multi-class problem, a one-vs-rest approach was taken, where for each level of obesity, separate binary classifiers were created. The *liblinear* solver was used, which is suitable for multi-class classification, and also, *max_iter* was set to 2000 to ensure convergence.

Test Set Performance:

- **Accuracy**: 58.5%
- **Top-2 Accuracy**: 73.6%

- **Confusion Matrix**:

$$\begin{bmatrix} 20 & 3 & 3 & 3 & 0 & 2 & 0 \\ 9 & 4 & 5 & 4 & 3 & 4 & 6 \\ 0 & 1 & 25 & 7 & 2 & 2 & 1 \\ 0 & 0 & 0 & 34 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 30 & 0 & 0 \\ 3 & 3 & 4 & 3 & 1 & 6 & 2 \\ 2 & 1 & 5 & 2 & 5 & 1 & 5 \end{bmatrix}$$



- **Mean Accuracy (5-fold CV)**: 54.6%
- **Cross-Validation Scores**: [42.3%, 60.4%, 54.5%, 55.2%, 60.4%]

```
logreg = LogisticRegression(max_iter=2000, solver='liblinear')
logreg.fit(X_train, y_train)
logreg_cv_scores = cross_val_score(logreg, X, y, cv=5, scoring='accuracy')
logreg_cv_mean = logreg_cv_scores.mean()
accuracy_logreg = accuracy_score(y_test, logreg.predict(X_test))
confusion_logreg = confusion_matrix(y_test, logreg.predict(X_test))
unique_labels = np.unique(y_train)
top_k_logreg = top_k_accuracy_score(y_test, logreg.predict_proba(X_test), k=2, labels=unique_labels)
```

Analysis:

- **Boundary Classes Performance**: It provided comparatively good performance on the extreme classes, like *Normal Weight* (Class 1) and *Obesity Type II* (Class 5). Most of the samples were correctly classified for *Obesity Type II* without any confusion, indicating that the model is effective in identifying extreme levels of obesity where the values of the features involved are quite distinctive.
- **Misclassifications in Intermediate Classes**: Indeed, the greatest confusion arose in classifying between consecutive levels, for example, *Overweight Level I* (Class 2) and *Obesity Type I* (Class 4). These two classes share borderline lifestyle patterns leading to their very common misclassification. This can also be seen from the confusion matrix, where many of the samples labeled as *Overweight Level I* were incorrectly classified into the other categories. This may also indicate that Logistic Regression struggled to find clear decision boundaries between such intermediate classes.
- **Overfitting to Dominant Patterns**: Logistic regression seems to be fitting on some big patterns, such as identifying extremes of obesity, but linearity limits the model's ability to handle subtle differences between intermediate categories of obesity. Probably due to this fact, the Top-2 accuracy, at 73.6%, is far superior compared to the overall accuracy at 58.5%, with the correct class often ranking as the second-best prediction, hinting at the limited power of the model to fully discern similar classes.
- **Cross-Validation Variability**: The scores of cross-validation are in the range of 42.3% - 60.4%, which is evidence that the model predictions are very sensitive with respect to a particular data split. This variability means the model is not able to generalize well across similar feature values from different classes.

Overall, while Logistic Regression provides a good baseline for performance, the model requires linear decision boundaries, which are inappropriate for the complex patterns present in this dataset. While this works well when classes are well-separated, it performs rather poorly with overlapping obesity categories. Thus, the high misclassification rate was found for the intermediate levels of obesity.

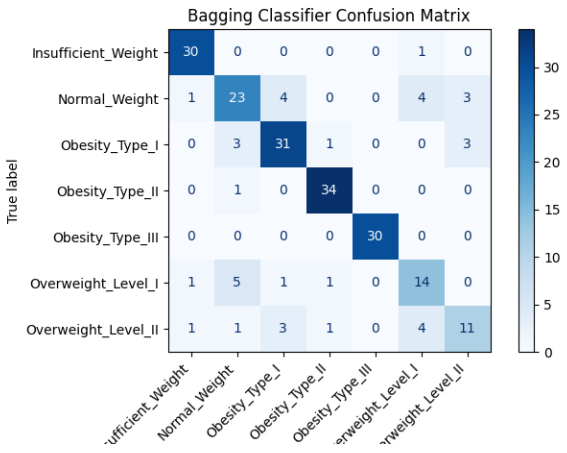
2.2 Bagging Classifier

Bagging is an ensemble method that works on improving model performance by training a set of weak learners on different subsets of the training data and averaging their predictions. Herein, the Bagging Classifier was done with the base estimator as Decision Trees. It would help in reducing overfitting and variance because the results of multiple decision trees would be averaged out; therefore, generally improving the generalization performance on unseen data.

Test Set Performance:

- **Accuracy**: 81.6%
- **Top-2 Accuracy**: 90.1%
- **Confusion Matrix**:

$$\begin{bmatrix} 30 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 23 & 4 & 0 & 0 & 4 & 3 \\ 0 & 3 & 31 & 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 34 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 30 & 0 & 0 \\ 1 & 5 & 1 & 1 & 0 & 14 & 0 \\ 1 & 1 & 3 & 1 & 0 & 4 & 11 \end{bmatrix}$$



- **Mean Accuracy (5-fold CV)**: 80.5%
- **Cross-Validation Scores**: [56.3%, 85.5%, 85.5%, 83.4%, 91.7%]

```
bagging = BaggingClassifier(n_estimators=50, random_state=42)
bagging.fit(X_train, y_train)
bagging_cv_scores = cross_val_score(bagging, X, y, cv=5, scoring='accuracy')
bagging_cv_mean = bagging_cv_scores.mean()
accuracy_bagging = accuracy_score(y_test, bagging.predict(X_test))
confusion_bagging = confusion_matrix(y_test, bagging.predict(X_test))
top_k_bagging = top_k_accuracy_score(y_test, bagging.predict_proba(X_test), k=2, labels=unique_labels)
```

Analysis:

- **Strength in Identifying Normal and Extreme Classes**: The classifier performed really well with Normal Weight and Obesity Type II classes, both of which were correctly predicted. This is indicative of the fact that this ensembling approach very well creates separating patterns between a healthy individual and a person at an extreme level of obesity. The variance among individual trees of Bagging reduces due to its ensemble nature, making it generalize well for these categories.
- **Handling Intermediate Classes with Moderate Success**: While the model worked better on the intermediate classes compared to Logistic Regression, some confusion between Overweight Level I and Obesity Type I is still evident. The existence of similar lifestyle attributes in both classes makes separation even challenging for an ensemble model to precisely find. However, the Bagging Classifier outperformed Logistic Regression by correctly predicting the majority of samples within those classes, as depicted by its accuracy of 81.6%.
- **Top-2 Accuracy Reflects Robustness**: The Top-2 accuracy was 90.1%, showing that even for the wrong predictions, in most cases the true class was ranked second. This may be crucially important in practical usage because the near-miss predictions can still provide useful insights.
- **Consistent Cross-Validation Performance**: The score in cross-validation oscillates between 56.3% and 91.7%, with an average of 80.5%. The similarity shows that the Bagging Classifier has generalized quite well across the different folds, meaning that the resulting ensemble model will be less sensitive to splits between train and test, which is expected when comparing it to Logistic Regression. This stability underlines a strength of Bagging on datasets with overlapping or noisy features.

With the Bagging Classifier, it became clear that these algorithms befit complex pattern datasets with even overlapped classes. While this is far from ideal, there is room for improvement in the separation of the obesity level intermediates; it gives a rough idea that the general performance of the Bagging Classifier is a robust and reliable model in order to predict the level of obesity.

4. Comparison

The complexity for the task of predicting obesity levels was given by the contrast between the Logistic Regression and the Bagging Classifier models. Logistic Regression was useful for the baseline but was limited due to its linearity, reaching an accuracy of 58.5%, mostly failing in the middle categories of obesity. However, the Bagging Classifier worked well in capturing the non-linear pattern of this problem and reached an accuracy of 81.6%, with more consistent cross-validation results.

The key difference is that Bagging works better for overlapping or complex feature data since aggregating over multiple decision trees reduces overfitting. On the other hand, although very efficient and interpretable, Logistic Regression failed to generalize very well for all categories of obesity, especially for classes whose feature values lay quite close to those of another class. This is captured in Top-2 accuracy: Logistic Regression had 73.6%, while Bagging achieved 90.1%, showing that Bagging is better at highly ranking the correct class.

Cross-validation results again emphasize Bagging's robustness since the scores are invariably close to the average, at 80.5%, while, for example, Logistic Regression ranges between 42.3% and 60.4%. This suggests that Bagging is less sensitive to whatever variation may take place in the training data and will thus yield, from a practical perspective, a more reliable model.

In all, though Logistic Regression is valuable as a baseline model, Bagging Classifier creates better generalization and accuracy, especially in complex pattern and non-linear relationship datasets.

5. Conclusion

This analysis studied the prediction of the levels of obesity based on health and lifestyle attributes using two different machine learning models, namely, Logistic Regression and the Bagging Classifier. The dataset captured various behavioral patterns relating to eating habits, physical activity, and family history of overweight, which were critical in understanding obesity. However, the classes corresponding to adjacent categories of obesity, for example, Overweight Level I and Obesity Type I, were not well-separated.

Results have shown that Logistic Regression is clearly crippled by its linearity, with only 58.5% test accuracy, and it struggles most with the classes in the middle, which are very similar in pattern, for example, irregular exercise and overeating. This indication of sensitivity to the split of data is further supported by the variability in cross-validation results from 42.3% to 60.4%, hence not robust on overlapping feature values. While the top-2 accuracy of 73.6% showed that very often the correct class was ranked second, it is not the best model to capture the complex patterns inherent in the dataset.

In contrast, the Bagging Classifier performed rather stronger, reaching 81.6% on the test set, while the Top-2 accuracy was equal to 90.1%. The ensemble allowed the model to cope better with non-linear relationships and reduce variance, hence providing more stable predictions. The mean cross-validation accuracy of 80.5% reflects that the scores across folds are rather consistent, and this signals good generalization of the Bagging Classifier even for such overlapping classes.

Insights from the Data: The models point out an intrinsic difficulty in making predictions on obesity levels based on lifestyle attributes, especially for intermediate categories of obesity. Although the dataset proposes meaningful indicators of physical activity and eating habits, people from different neighboring classes actually present overlapped behaviors, making it tough to draw boundaries. The extreme levels of obesity, like Obesity Type II, had been easier to categorize and had probably developed due to more distinctive patterns occurring in the data, such as high caloric intake and low activity levels.

Model Comparison: Overall, the Bagging Classifier can learn those patterns way better compared to Logistic Regression, which scores significantly lower. Its capability for ensemble learning by reducing overfitting easily qualified Bagging for this task. As for Logistic Regression, though interpretable and fast, it cannot cope with the intricacies of the current dataset and should be regarded more as a baseline than a solution.

References

[1] UCI Machine Learning Repository, *Estimation of Obesity Levels Based On Eating Habits and Physical Condition*, 2019, `https://doi.org/10.24432/C5H31Z`.

This project aims to predict obesity levels based on personal and lifestyle attributes using machine learning techniques, specifically XGBoost and k-Nearest Neighbors (k-NN). By focusing on factors such as age, gender, family history of overweight, smoking habits, and physical activity frequency, we aim to develop a reliable model for assessing obesity risk. Notably, height and weight variables were excluded to simplify the model and avoid over-reliance on these common indicators, allowing us to explore more nuanced lifestyle factors.

Dataset Overview: The dataset includes features such as:

  - **Demographics**: Age, Gender
  - **Health and Lifestyle**: Family History of Overweight (FamilyHistory), Frequent Consumption of High-Calorie Food (FAVC), Daily Meals (NCP), Smoking Status (Smoke), and Physical Activity Frequency (FAF).

- **Problem Setup**: This is a classification problem, with the target variable being the obesity level category.
- **Data Splitting**: The dataset was split into 90% training and 10% test sets to evaluate model performance on unseen data.

Data Preparation

- **Selected Features**: For model simplicity and relevance, we focused on features closely linked to obesity, such as Age, Gender, and indicators like food consumption habits and physical activity.
- **Preprocessing Steps**:
  - **Encoding Categorical Variables**: Label encoding was applied to variables like Gender, FamilyHistory, FAVC, and Transport to convert them into numeric values for model training.
  - **Normalization and Standardization**: We used MinMaxScaler for normalization and StandardScaler for standardization. This combined scaling was essential for optimizing both XGBoost and k-NN performance, as k-NN is especially sensitive to feature scaling.

Model Development

- **XGBoost Model**
  - **Configuration**: An XGBoost classifier was used with multi:softmax for multi-class classification, num_class for the unique obesity levels, and n_estimators set to 50.

    - **Training Process**: The model was trained on scaled training data, where each iteration refined predictions based on previous errors, capturing complex relationships between features.

- **k-Nearest Neighbors (k-NN) Model**
    - **Configuration**: The k-NN model was set with n_neighbors=5, chosen based on preliminary cross-validation. This configuration accounts for the nearest five data points in prediction, balancing between bias and variance.
    - **Training Process**: Trained on standardized data, k-NN leverages scaled feature distances, relying on normalized and standardized data for accurate predictions.
- **Code Snapshot**: Below is a sample of the Python code for model training.

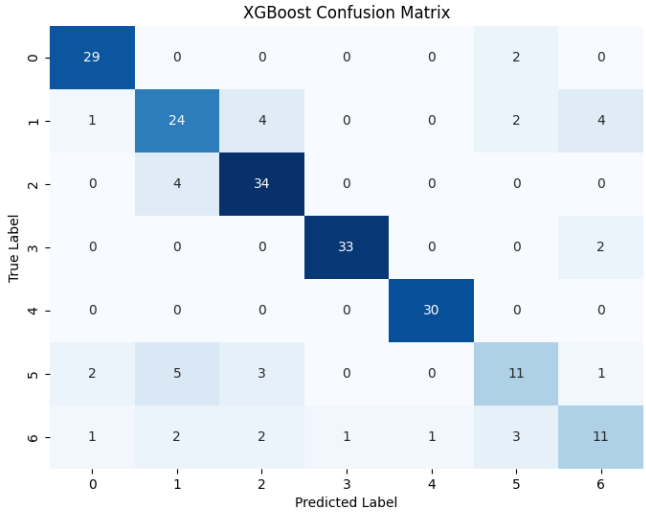
```
# Evaluating XGBoost model
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
report_xgb = classification_report(y_test, y_pred_xgb)
print("XGBoost Accuracy:", accuracy_xgb)
print("Classification Report for XGBoost:\n", report_xgb)

# Evaluating k-NN model
accuracy_knn = accuracy_score(y_test, y_pred_knn)
report_knn = classification_report(y_test, y_pred_knn)
print("k-NN Accuracy:", accuracy_knn)
print("Classification Report for k-NN:\n", report_knn)
```

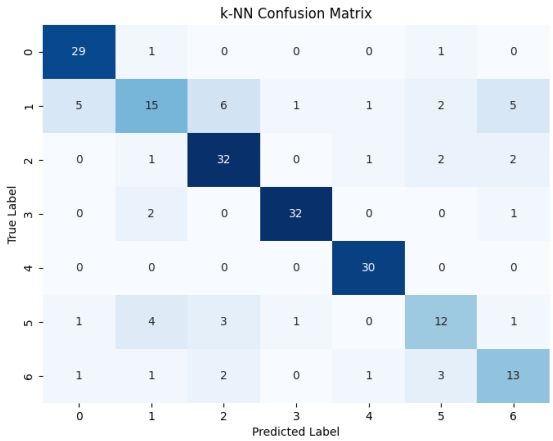
Model Evaluation and Results

- **XGBoost Model Evaluation**
    - **Accuracy**: Achieved an accuracy of 81.1%, reflecting strong predictive performance across obesity categories.
    - **Top-3 Accuracy**: The model achieved a Top-3 accuracy of 92.9%, indicating high reliability in multi-class predictions.
    - **Classification Report**: Precision, recall, and F1-score were high across classes, although minor variation appeared due to data distribution.
    - **Confusion Matrix**: The confusion matrix (Figure 1) demonstrates high accuracy across classes, with occasional misclassifications, particularly

in classes 1 and 5.



- **k-NN Model Evaluation**
  - **Accuracy**: The k-NN model attained an accuracy of 76.9%, slightly lower than XGBoost but acceptable.
  - **Top-3 Accuracy**: Top-3 accuracy was 86.3%, lower than XGBoost, reflecting its dependency on accurate scaling for class differentiation.
  - **Classification Report**: Precision, recall, and F1-score remained reasonably high, though lower in ambiguous classes, such as class 1.
  - **Confusion Matrix**: The k-NN confusion matrix (Figure 2) shows some misclassifications, especially between similar classes like 1 and 2, likely due to overlapping feature distributions.



- **Code Snapshot for Evaluation**: Below is the code snippet used for model evaluation. *(Insert snapshot here)*
- `python`

```
# Evaluating XGBoost model
```

```
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
report_xgb = classification_report(y_test, y_pred_xgb)
print("XGBoost Accuracy:", accuracy_xgb)
print("Classification Report for XGBoost:\n", report_xgb)
# Evaluating k-NN model
accuracy_knn = accuracy_score(y_test, y_pred_knn)
report_knn = classification_report(y_test, y_pred_knn)
print("k-NN Accuracy:", accuracy_knn)
print("Classification Report for k-NN:\n", report_knn)
```

Discussion, Limitations, and Future Improvements

- **Model Performance Comparison**: The XGBoost model outperformed k-NN in both accuracy and Top-3 accuracy, showing it better captures complex relationships between obesity indicators. XGBoost's ensemble approach enables it to recognize these patterns more effectively than k-NN's distance-based approach, which can struggle with overlapping features.
- **Observations**:
    - XGBoost achieved high precision and recall, especially in well-represented classes.
    - k-NN was effective in defined classes but less accurate in overlapping categories, illustrating its limitations in high-dimensional space without precise feature scaling.

Limitations and Potential Improvements

- **Class Imbalance**: Some obesity categories had fewer samples, impacting prediction accuracy in these classes. Future improvements might involve re-sampling techniques like SMOTE for better class balance.
- **Hyperparameter Tuning**: Additional tuning could improve results. Increasing n_estimators in XGBoost and experimenting with different distance metrics in k-NN could enhance accuracy.
- **Feature Engineering**: Adding additional obesity indicators, such as metabolic rate estimates or body measurements, could improve the model's predictive power.

- **Data Augmentation**: Expanding the dataset, potentially with related obesity datasets, might further enhance both models by providing broader representation across demographic groups.

Conclusion

- In summary, XGBoost provided better accuracy and robustness than k-NN for predicting obesity levels based on personal and lifestyle attributes. The results indicate that ensemble methods like XGBoost are more suitable for complex classification tasks, while k-NN benefits more from high-dimensional data with clear feature scaling. With further enhancements, these models could significantly contribute to public health efforts by identifying obesity risk factors early.

Reference: *UCI Machine Learning Repository*. (2019, August 26). Archive.ics.uci.edu. https://archive.ics.uci.edu/dataset/544/estimation+of+obesity+levels+based+on+eating+habits+and+physical+condition

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
import xgboost as xgb
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, top_k_accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
df = pd.read_csv('/Users/zhougary/Desktop/obs1.csv')

# Selecting relevant columns for analysis
columns_to_use = [
    'Age', 'Gender', 'family_history_with_overweight', 'FAVC', 'FCVC', 'NCP', 'CAEC', 'SMOKE',
    'CH2O', 'SCC', 'FAF', 'TUE', 'CALC', 'MTRANS', 'NObeyesdad'
]

# Extracting the relevant data
df = df[columns_to_use].copy()

# Rename columns for easier handling
df.columns = ['Age', 'Gender', 'FamilyHistory', 'FAVC', 'FCVC', 'NCP', 'CAEC', 'Smoke',
              'CH2O', 'SCC', 'FAF', 'TUE', 'Calc', 'Transport', 'Obesity']

# Encoding categorical variables
le_gender = LabelEncoder()
df['Gender'] = le_gender.fit_transform(df['Gender'])

le_family = LabelEncoder()
df['FamilyHistory'] = le_family.fit_transform(df['FamilyHistory'])

le_favc = LabelEncoder()
df['FAVC'] = le_favc.fit_transform(df['FAVC'])

le_caec = LabelEncoder()
df['CAEC'] = le_caec.fit_transform(df['CAEC'])

le_smoke = LabelEncoder()
df['Smoke'] = le_smoke.fit_transform(df['Smoke'])

le_scc = LabelEncoder()
df['SCC'] = le_scc.fit_transform(df['SCC'])

le_calc = LabelEncoder()
df['Calc'] = le_calc.fit_transform(df['Calc'])

le_transport = LabelEncoder()
df['Transport'] = le_transport.fit_transform(df['Transport'])

le_obesity = LabelEncoder()
df['Obesity'] = le_obesity.fit_transform(df['Obesity'])

# Splitting data into features and target
X = df.drop('Obesity', axis=1)
y = df['Obesity']

# Splitting into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)

# Normalizing the features using Min-Max normalization
scaler = MinMaxScaler()
X_train_normalized = scaler.fit_transform(X_train)
X_test_normalized = scaler.transform(X_test)

# Standardizing the features
scaler_standard = StandardScaler()
X_train_scaled = scaler_standard.fit_transform(X_train_normalized)
X_test_scaled = scaler_standard.transform(X_test_normalized)

```

```python

# Building the XGBoost model
xgboost_model = xgb.XGBClassifier(objective='multi:softmax', num_class=len(df['Obesity'].unique()), n_estimators=50, random_state=42)
xgboost_model.fit(X_train_scaled, y_train)
y_pred_xgb = xgboost_model.predict(X_test_scaled)

# Evaluating the XGBoost model
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
report_xgb = classification_report(y_test, y_pred_xgb)

# XGBoost Top-3 Accuracy
y_proba_xgb = xgboost_model.predict_proba(X_test_scaled)
top3_accuracy_xgb = top_k_accuracy_score(y_test, y_proba_xgb, k=2)

# Building the k-NN model
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train_scaled, y_train)
y_pred_knn = knn_model.predict(X_test_scaled)

# Evaluating the k-NN model
accuracy_knn = accuracy_score(y_test, y_pred_knn)
report_knn = classification_report(y_test, y_pred_knn)

# k-NN Top-3 Accuracy
y_proba_knn = knn_model.predict_proba(X_test_scaled)
top3_accuracy_knn = top_k_accuracy_score(y_test, y_proba_knn, k=2)

# Outputting results
print("XGBoost Accuracy:", accuracy_xgb)
print("XGBoost Classification Report:\n", report_xgb)
print("XGBoost Top-3 Accuracy:", top3_accuracy_xgb)

print("k-NN Accuracy:", accuracy_knn)
print("k-NN Classification Report:\n", report_knn)
print("k-NN Top-3 Accuracy:", top3_accuracy_knn)

# Function to plot confusion matrix
def plot_confusion_matrix(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title(title)
    plt.show()

# Plotting confusion matrix for XGBoost model
print("Confusion Matrix for XGBoost Model")
plot_confusion_matrix(y_test, y_pred_xgb, title="XGBoost Confusion Matrix")

# Plotting confusion matrix for k-NN model
print("Confusion Matrix for k-NN Model")
plot_confusion_matrix(y_test, y_pred_knn, title="k-NN Confusion Matrix")

```

```
XGBoost Accuracy: 0.8113207547169812
XGBoost Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.94      0.91        31
           1       0.69      0.69      0.69        35
           2       0.79      0.89      0.84        38
           3       0.97      0.94      0.96        35
           4       0.97      1.00      0.98        30
           5       0.61      0.50      0.55        22
           6       0.61      0.52      0.56        21

    accuracy                           0.81       212
   macro avg       0.79      0.78      0.78       212
weighted avg       0.80      0.81      0.81       212

XGBoost Top-3 Accuracy: 0.9292452830188679
k-NN Accuracy: 0.7688679245283019
k-NN Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.94      0.87        31
           1       0.62      0.43      0.51        35
           2       0.74      0.84      0.79        38
           3       0.94      0.91      0.93        35
           4       0.91      1.00      0.95        30
           5       0.60      0.55      0.57        22
           6       0.59      0.62      0.60        21

    accuracy                           0.77       212
   macro avg       0.75      0.75      0.75       212
weighted avg       0.76      0.77      0.76       212

k-NN Top-3 Accuracy: 0.8632075471698113
Confusion Matrix for XGBoost Model
```

Random forest and Naive Bayes modeling

Data cleaning: Duplicate/missing/odd values were identified. 24 duplicates were removed. This was only done after data was split into test/train/validate sets to ensure everyone had the same data in each set.
Preprocessing: The dataset had 499 real responses and 1612 entries generated with the SMOTE filter in Weka to balance the number of responses in each obesity category (Palechor, & la Hoz Manotas, 2019a). The synthetic data had continuous ranges for ordinal data, which was converted back to ordinal with a K-Bins Discretizer with uniform bins. This allows the model to reflect real world data more closely & is the expected data for the Categorical Naive Bayes model. Other ordinal features were labeled with numbers (including the obesity level), maintaining ordering by specifying the categories explicitly. I changed the data type to numpy integer to help differentiate the data, scikit-learn uses floating point for its calculations. One-hot encoding was used with the nominal feature (transport type). Since gender had two recorded values, labeling was used as it is equivalent to dummy encoding.

clean.py model.py

```
# synthetic data as floating point. Use equal size bins.
for column in data.columns[4:]:
    if data[column].dtype == np.float64:
        n_categories = int(data[column].max() - data[column].min() + 1)
        bins = KBinsDiscretizer(n_bins=n_categories, encode="ordinal", strategy="uniform")
        data[column] = bins.fit_transform(data[[column]])
        data[column] = data[column].convert_dtypes()
```

clean.py model.py

```
bin_encoder = OrdinalEncoder(categories=[["no", "yes"]],
                             dtype=np.int8)
for column in columns_binary:
    data[column] = bin_encoder.fit_transform(data[[column]])
```

clean.py model.py

```
# One-hot encoding of transport type, as it lacks natural ordering
# towardsdatascience.com/categorical-encoding-using-label-encoding-and-one-hot-encoder-911ef77fb5bd
oh_encoder = OneHotEncoder(dtype=np.int8, sparse_output=False)
transport_data = pd.DataFrame(oh_encoder.fit_transform(data[["MTRANS"]]))
transport_data.columns = list(oh_encoder.categories_[0])
data = data.join(transport_data)

# Reorder columns so obesity is last
column_order = list(data.columns)[:-1 - len(transport_data.columns)]
column_order.extend(list(transport_data.columns))
column_order.append("NObeyesdad")
data = data[column_order]
```

Age was not scaled as a random forest model is scaling invariant, and a Categorical Naive Bayes model assumes ordinal data so Age was excluded.
Weight and height were excluded to avoid overfitting (as obesity was calculated based on the Mexican adjusted BMI index). All other features were used to predict obesity levels. After preprocessing, there were 18 predictors (17 when excluding age for the Categorical Naive Bayes). Data was split into train, validate, and test data sets (81%, 9%, 10% split respectively). Test data was only evaluated once the models were finalised.

Evaluation code: The evaluation code was shared between models. I passed either the validation or test data to the `evaluateModel` function and let the model predict the obesity level & probabilities for each obesity level. These were passed to the `confusion_matrix`, `top_k_accuracy_score`, & `ConfusionMatrixDisplay` functions.

clean.py model.py

```python
def evaluateModel(model, X, y_true, model_name, y_labels):
    y_pred_proba = model.predict_proba(X)
    y_pred = model.predict(X)
    print("top_k_accuracy_score", metrics.top_k_accuracy_score(y_true, y_pred_proba))
    print("Confusion matrix:", metrics.confusion_matrix(y_true, y_pred))
    disp = metrics.ConfusionMatrixDisplay.from_estimator(model,
                                                         X,
                                                         y_true,
                                                         display_labels=y_labels,
                                                         xticks_rotation="vertical",
                                                         )
    disp.plot()
    plt.xticks(rotation=90)
    plt.title(model_name)
    plt.tight_layout()
    plt.savefig(model_name + ".png", dpi=300)
```

Both models used grid searches to find the best hyperparameters, which were also printed out.

model.py clean.py

```python
print("Best hyperparameters", grid_clf.best_params_)
```

Random forest model

Model choice: The random forest model handles categorical data well. Since it uses a number of Decision Trees that only have some of the data, it avoids overfitting. While it is harder to interpret than decision trees (we can't follow a decision flowchart), the random forest does indicate the importance of different features in the model. The random forest also deals with non-linear data, useful as we suspect the variables interact non-linearly (scikit-learn developers, 2024b).

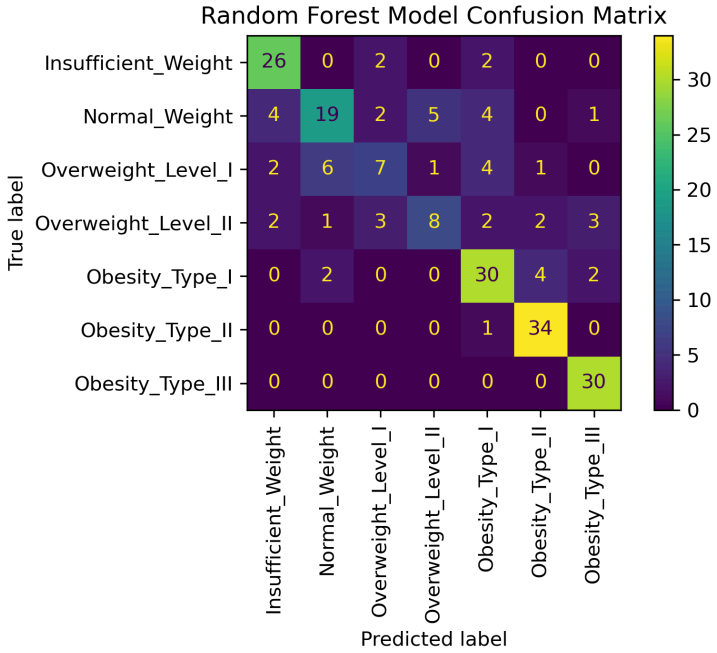
Model Production: A Random Forest Classifier was created with a random seed for repeatability and passed to a grid search to select the best hyperparameters (number of decision trees, maximum depth of each tree and sample size given to each tree). The grid search uses 5-fold cross-validation to avoid overfitting. Maximum depth was limited to 6 to further minimise the risk of overfitting. The algorithm was then run by calling the `fit` method on the training data. The choices were made after experimenting with a decision tree (which overfitted), then Random Forest (which was then optimised with the grid search). Models were evaluated with the validation data set during experimentation.

clean.py model.py

```python
forest_clf = RandomForestClassifier(random_state=SEED)
param_grid = {
    "n_estimators": [50, 100, 200, 800],
    "max_depth": [3, 4, 5, 6],
    "max_samples": [0.1, 0.3, 0.5, 0.6, 0.8, 0.9],
}
grid_clf = GridSearchCV(forest_clf,
                        param_grid).fit(X_train, y_train)
```

Evaluation: The model performs well on the test data. The Top-2 accuracy score (0.85) was moderately high: 85% of all predictions had the correct obesity level as 1 of the 2 most likely obesity levels (of 7). The confusion matrix shows mispredictions are usually only a couple categories off, and the model does better for more obese individuals. However, it still performs worse for less obese levels, and has greater spread in miscategorisation. This is likely because more obese individuals are more likely to have similar behavior, which the decision tree is able to predict, while there is greater variability for overweight and normal weight individuals. Since the obesity level is determined purely by BMI, some marked as overweight may not truly have an unhealthy weight or unhealthy habits (e.g. bodybuilder), causing the

random forest to fail to correctly classify them (though with more data and a model with greater decision tree depth may identify the BMI correctly). This model is unlikely to be overfit and wasn't cherry-picked: the validation data had a similar top-2 accuracy score of 0.88, and many precautions were used.



The 5 features with the greatest importance to the model, in order, were age (0.19%), gender (0.14%), frequency of consumption of vegetables (10%), family history of being overweight (10%), and consumption of food between meals (9%). Surprisingly, physical activity did not have as strong an influence as an individual's eating habits. This may be due to the imprecision of the data (e.g. only one question about physical activity frequency, and caloric consumption monitoring had little variability).

clean.py model.py

```
142 print("Best hyperparameters", grid_clf.best_params_)
143 print("Feature importance")
144 grid_feature_importance = pd.DataFrame(grid_clf.best_estimator_.feature_importances_,
145                                        index=input_names)
146 print(grid_feature_importance.sort_values(0, ascending=False))
```

The final hyperparameters selected by the grid search were a maximum decision tree depth of 6, sample size of 60% of the training data set, and 200 decision trees for the random forest model.

Categorical Naive Bayes model:

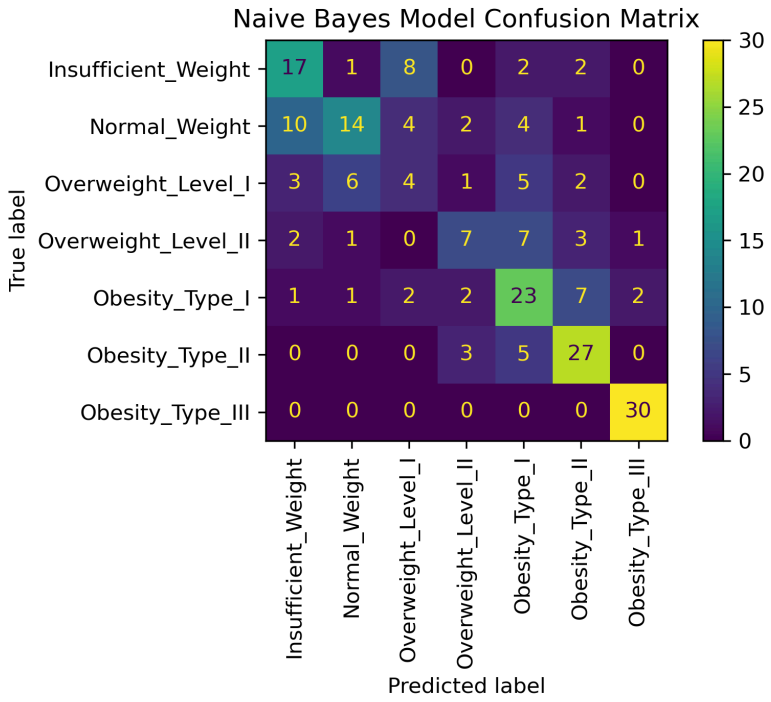
Model Choice: although the Bayes model assumes features are independent (and makes predictions using Bayes theorem), it tends to perform well even when the assumption is violated. The categorical Naive Bayes model assumes categorical variables, which makes it suited to this mostly categorical dataset. It is a quicker model to train than the Random Forest model. After balancing with synthetic data, this model is suitable even without smoothing (scikit-learn developers, 2024a, MLNerds, 2021).

Model Production: A similar grid search was used for the Naive Bayes model, testing the only hyperparameter (alpha for smoothing). The category count was passed to the Naive Bayes model to avoid issues with samples where not all values were present in the 5-fold cross validation. The training was done on the training data using the `fit` method.

clean.py model.py

```
161 alpha_search = {
162 "alpha": [(p+1)/10.0 for p in range(9)]
163 }
164
165 category_count = [len(data[name].unique()) for name in input_names if name != "Age"]
166
167 bayes_clf = GridSearchCV(CategoricalNB(min_categories=category_count),
168                          alpha_search).fit(X_train, y_train)
```

Evaluation: The best hyperparameter choice was `alpha=0.1`, which produced a model with a Top-2 accuracy score of 0.77 on the test data. From the confusion matrix, it is generally worse at predicting the exact obesity level; however it is not significantly more likely to mispredict obesity by several categories (with the exception of those with insufficient weight). Regardless, it is a worse model than the Random Forest model for this dataset, however it is not that much worse. The difference in performance may be because of the lack of independence between features. Comparing results to the validation data set (top-2 accuracy score of 0.80), and not performance is not too good, the data does not suffer from overfitting.



Conclusion

All models performed well under the performance metrics considered. The confusion matrices show that they perform better for more extreme obesity levels, but even when they make incorrect predictions they are only off by a class. This is true of all models considered. They are all potentially suitable for predictions of obesity level; however, methods that deal with nonlinearity tend to perform better.

Model	Logistic Regression	Naive Bayes	SVM	Random Forest	kNN (n=5)	MLP	Bagging Classifier	XGBoost
Top-2 accuracy score	0.74	0.77	0.81	0.85	0.86	0.87	0.90	0.93

Logistic regression: This method performed the worst, though it is still quite good at predicting the correct category, placing it in the top 2 most likely 74% of the time. This method is able to handle multi-class predictions but uses a linear model so may not be able to identify non-linear relationships between features and the obesity level. It is better suited to problems with linear decision boundaries.
Categorical Naive Bayes: While this method is also passable, it performs relatively poorly and is unlikely to be used for this problem. It may not be useful if future data is composed of a mix of quantitative and categorical data. While it is designed to handle categorical data, it may be failing as features may not be independent. It is true that the model often performs well even when the assumption of independence is broken, this is not guaranteed. This method is often used as it is simple to understand (it uses Bayes theorem to calculate probabilities), and training is not memory/time intensive. However, these advantages are not as useful in identifying behaviours that are linked to obesity: we don't need frequent training and its top-2 accuracy is not the best.
Random Forest: Discussed later.
Support Vector Machine (SVM): SVMs work well when there is a clear margin of separation between classes, allowing them to produce accurate classifications. This may explain why they don't work well in the obesity dataset. They perform effectively in high-dimensional spaces and are particularly useful in cases where the number of features exceeds the number of samples, so may be useful if a more indepth survey is conducted. SVMs also have good memory efficiency, using only a subset of training data (support vectors) to define the decision boundary, leading to a sparse solution that prevents overfitting. Their ability to model non-linear decision boundaries through the kernel trick adds to their versatility, making them suitable for both classification and regression tasks. Additionally, SVMs are robust to noise, as the decision boundary is primarily influenced by the closest points to the boundary, and they exhibit strong generalisation performance with new, unseen data.
Despite these advantages, SVMs also have some drawbacks. They can be computationally expensive, especially with large datasets, as solving the quadratic optimization problem requires significant time and resources. SVMs can also be memory-intensive, particularly when the kernel matrix becomes large. Choosing the right kernel and tuning parameters such as the regularisation parameter can be difficult and has a significant impact on performance. Additionally, SVMs are primarily designed for two-class problems, and while they can handle multi-class problems using strategies like one-versus-one or one-versus-all, these approaches can be complex and less efficient. SVMs lack a probabilistic interpretation of their decision boundaries, which can limit their usefulness in applications requiring probability estimates.

k Nearest Neighbour (kNN): This performs better than other simple models mentioned above, and simply predicts based on the majority vote of the nearest neighbours. The model is easy to understand, however assumes that each obesity level forms separate clusters of behaviour. The good performance indicates that this is probably the case, however it is likely limited because feature patterns overlap significantly between classes.

Multilayer perceptrons (MLPs): MLPs have several strengths that make them versatile and effective in various applications. They can handle different types of input data, including both continuous and categorical variables, and can easily manage missing data. MLPs generalise well to new, unseen data when trained properly, which enhances their usefulness in real-world scenarios. Their scalability is another advantage, as adding hidden layers or nodes can improve performance on complex tasks. Additionally, MLPs excel in nonlinear modelling. MLPs can capture intricate relationships between inputs and outputs, making them suitable for a range of complex problems including predicting obesity levels.

However, MLPs come with notable weaknesses. They are considered "black box" models, meaning that the internal decision-making process is not easily interpretable, which can be a drawback in applications requiring transparency, such as justifying why behaviours are related to obesity classes. MLPs are also prone to overfitting, especially when the model is too complex or the training data is insufficient. Training an MLP can be time taking and computationally intensive, particularly with large datasets or deep networks. Furthermore, optimising the model's hyperparameters, such as the number of nodes, layers, activation function, and learning rate, is essential for achieving optimal performance but can be challenging and time-consuming.

Ensemble methods: The best models were ensemble methods based on decision trees: Gradient Boosting (XGBoost) and a Bagging Classifier. Random forest also did well but was relatively worse. This is expected as XGBoost uses a regularisation term to quantify the decision tree complexity and iteratively improves its model as each tree is added (XGBoost developers, 2022), whilst a Random Forest simply does a majority vote based on the predictions from an ensemble of trees. XGB's iterative approach also consumes fewer resources (2024b, September). While these ensemble methods use weak decision trees, the Bagging Classifier is able to work with more complex trees without overfitting (scikit-learn developers, 2024b). In contrast, the Random forest used a maximum of roughly 4 features per tree (while this can be adjusted, it works best with limited decision trees).

Feature importance: These models are more opaque than the decision trees they are based on, though they are still able to reveal the feature importance. As seen in the random forest model, the most important features were age (0.19%), gender (0.14%), frequency of consumption of vegetables (10%), family history of being overweight (10%), and consumption of food between meals (9%). The strong importance of gender may simply be because BMI does not account for gender or may indicate real differences (Gough Courtney & Carroll, 2022). Surprisingly, variables concerning physical activity were not the most important, possibly because of the coarseness of the survey, with only one question about physical activity frequency, and little variability in response to caloric consumption monitoring. A more detailed survey, or one with more accurate measures of obesity level would likely improve the results of these models.

Conclusion: For the task of identifying obesity levels and the behaviours that are related to obesity, XGBoost is preferred due to its top-2 accuracy score, efficiency, and ability to provide feature importance. It would still be usable if a more detailed survey with more quantitative data is collected. Despite its complexity, it also avoids overfitting.

References

Gough Courtney, M., & Carroll, A. (2022). Sex differences in overweight and obesity among Mexican Americans in the National Health and Nutrition Examination Survey: A comparison of measures. *SSM - population health*, *20*, 101297. https://doi.org/10.1016/j.ssmph.2022.101297

MLNerds. (2021, July 30). Naive Bayes Classifier : Advantages and Disadvantages . *Machine Learning Interviews*. https://machinelearninginterview.com/topics/machine-learning/naive-bayes-classifier-advantages-and-disadvantages/

Palechor, F. M., & la Hoz Manotas, A. de. (2019a). Dataset for estimation of obesity levels based on eating habits and physical condition in individuals from Colombia, Peru and Mexico. *Data in Brief, 25*, 104344. https://10.1016/j.dib.2019.104344

Palechor, F. M., & la Hoz Manotas, A. de. (2019b). *Estimation of Obesity Levels Based On Eating Habits and Physical Condition* [Dataset]. (2019). UCI Machine Learning Repository. https://doi.org/10.24432/C5H31Z

scikit-learn developers. (2024a, September). *1.9. Naive Bayes - scikit-learn 1.5.2 documentation*. Scikit-learn. https://scikit-learn.org/1.5/modules/naive_bayes.html

scikit-learn developers. (2024b, September). *1.11. Ensembles: Gradient boosting, random forests, bagging, voting, stacking*. Scikit-learn. https://scikit-learn.org/1.5/modules/ensemble.html

XGBoost developers. (2022, August 15). *Introduction to Boosted Trees.* XGBoost. https://xgboost.readthedocs.io/en/stable/tutorials/model.html