**CS251 Intro to AI**

**Semester Project Report**

**Name:** Umer Khalid **RegNo**: 2022605 **Instructor:** Mam Nazia Shehzadi

**Predicting Obesity Levels in Mexico, Peru, and Colombia**

**Introduction:**

Obesity is a pressing global health concern associated with various physical and mental health issues. As its prevalence continues to rise, understanding the factors influencing obesity and developing effective prediction models become imperative. This report explores the prediction of obesity levels in individuals from Mexico, Peru, and Colombia based on their eating habits and physical condition using machine learning models.

**Dataset Description:**

The dataset contains 17 attributes and 2111 records, including features like gender, age, height, weight, family history with overweight, dietary habits, physical activity level, and transportation mode. The target variable is the obesity level, categorized into seven classes ranging from Insufficient Weight to Obesity Type III. The data was collected directly from users through a web platform, with 77% generated synthetically using the Weka tool and the SMOTE filter.

Gender: Feature, Categorical, "Gender"

Age : Feature, Continuous, "Age"

Height: Feature, Continuous

Weight: Feature Continuous

family\_history\_with\_overweight: Feature, Binary, " Has a family member suffered or suffers from overweight? "

FAVC : Feature, Binary, " Do you eat high caloric food frequently? "

FCVC : Feature, Integer, " Do you usually eat vegetables in your meals? "

NCP : Feature, Continuous, " How many main meals do you have daily? "

CAEC : Feature, Categorical, " Do you eat any food between meals? "

SMOKE : Feature, Binary, " Do you smoke? "

CH2O: Feature, Continuous, " How much water do you drink daily? "

SCC: Feature, Binary, " Do you monitor the calories you eat daily? "

FAF: Feature, Continuous, " How often do you have physical activity? "

TUE : Feature, Integer, " How much time do you use technological devices such as cell phone, videogames, television, computer and others? "

CALC : Feature, Categorical, " How often do you drink alcohol? "

MTRANS : Feature, Categorical, " Which transportation do you usually use? "

NObeyesdad : Target, Categorical, "Obesity level"

**Model Architectures:**

I have used FeedForward Neural Network which is a type of sequential modeling and it moves only in one direction and is best suited for tabular data, then Gated Recurrent Unit which is a type of Recurrent neural networks and is best suited for image processing image data but I have used it for categorical features and lastly ResNet which uses residual layers and is a type of convolutional neural networks and is also best suited for image processing , object detection but in our case used for categorical data.

**1. Hybrid Model: FNN and GRU**

Data Preprocessing: Categorical features were one-hot encoded, and numerical features were standardized.

**Model Architecture:** The hybrid model combines a Gated Recurrent Unit (GRU) for processing categorical features and a Feedforward Neural Network (FNN) for numerical features and then combine the results of both as input into another FNN and gets its output as a result for classification. This approach uses feature extracting from one model and then classification from another.

**Evaluation Metrics:** Accuracy, Precision, Recall, F1-score, AUC-ROC, Confusion Matrix.

**Results:** The hybrid model achieved an accuracy of 94.08%, with high precision, recall, and F1-score across all classes.

**2. Stacked Model: FNN and GRU**

**Model Architecture:** Similar to the first hybrid model, but with a stacked approach where the output of the first stage is fed again as input to the same model for classification.

**Evaluation Metrics:** Same as the hybrid model.

**Results:** The stacked model yielded comparable performance to the hybrid model, with an accuracy of 94.02% and similar precision, recall, and F1-score.

**3. Hybrid Model: ResNet and FNN**

**Model Architecture:** Combines a ResNet-like architecture for processing categorical features and an FNN for numerical features. This model also uses the approach used in the first hybrid model of feature extracting from one model and then classification from another.

**Evaluation Metrics:** Same as previous models.

**Results:** Achieved an accuracy of 94.02%, with consistent precision, recall, and F1-score.

**Code:**

**I have documented the code well enough in the comments**

First I have used data visualization to see relationships between different features and the target variable obesity level which is a categorical variable having 7 different classes ranging from insufficient weight to obesity level 3.

df.drop\_duplicates(inplace=True)

dfcat=df.select\_dtypes(exclude='number') # selecting only categorical features

dfcat.head()

import matplotlib.pyplot as plt

import seaborn as sns

# Set the style for seaborn plots

sns.set\_style("whitegrid")

# Plot : Bar chart for Gender

plt.figure(figsize=(8, 6))

sns.countplot(data=dfcat, x="Gender")

plt.title("Distribution of Gender")

plt.xlabel("Gender")

plt.ylabel("Count")

plt.legend(title="Weight Status")

plt.show()

A graph showing a distribution of gender

Description automatically generated

# Plot 1: Bar plot showing the distribution of weight status by gender

plt.figure(figsize=(8, 6))

sns.countplot(data=dfcat, x="Gender",hue='NObeyesdad')

plt.title("Distribution of weight status by Gender")

plt.xlabel("Gender")

plt.ylabel("Count")

plt.show()

# Plot 2: Stacked bar chart for family\_history\_with\_overweight and NObeyesdad

cross\_tab = pd.crosstab(dfcat["family\_history\_with\_overweight"], df["NObeyesdad"])

cross\_tab.plot(kind="bar", stacked=True, figsize=(10, 6))

plt.title("Distribution of Weight Status by Family History")

plt.xlabel("Family History with Overweight")

plt.ylabel("Count")

plt.xticks(rotation=0)

plt.legend(title="Weight Status")

plt.show()

# Plot 3: Bar plot showing the relationship between smoke and NObeyesdad

plt.figure(figsize=(8, 6))

sns.countplot(data=dfcat, x='SMOKE', hue="NObeyesdad")

plt.title("Distribution of Weight Status by SMOKE")

plt.xlabel("SMOKE")

plt.ylabel("Count")

plt.legend(title="Weight Status")

plt.show()

# Plot 4: Bar plot showing the relationship between CALC and NObeyesdad

plt.figure(figsize=(8, 6))

sns.countplot(data=dfcat, x='CALC', hue="NObeyesdad")

plt.title("Distribution of Weight Status by SMOKE")

plt.xlabel("CALC")

plt.ylabel("Count")

plt.legend(title="Weight Status")

plt.show()

# Plot 5: Bar plot showing the relationship between Gender and NObeyesdad

plt.figure(figsize=(8, 6))

sns.countplot(data=dfcat, x='MTRANS', hue="NObeyesdad")

plt.title("Distribution of Weight Status by MTRANS")

plt.xlabel('MTRANS')

plt.ylabel("Count")

plt.legend(title="Weight Status")

plt.show()

# Plot 6: Bar plot showing the relationship between FAVC and NObeyesdad

plt.figure(figsize=(8, 6))

sns.countplot(data=dfcat, x='FAVC', hue="NObeyesdad")

plt.title("Distribution of Weight Status by FAVC")

plt.xlabel("FAVC")

plt.ylabel("Count")

plt.legend(title="Weight Status")

plt.show()

# Plot 7: Bar plot showing the relationship between SCC and NObeyesdad

plt.figure(figsize=(8, 6))

sns.countplot(data=dfcat, x='SCC', hue="NObeyesdad")

plt.title("Distribution of Weight Status by SCC")

plt.xlabel("SCC")

plt.ylabel("Count")

plt.legend(title="Weight Status")

plt.show()

# Plot 8: Bar plot showing the relationship between CAEC and NObeyesdad

plt.figure(figsize=(8, 6))

sns.countplot(data=dfcat, x='CAEC', hue="NObeyesdad")

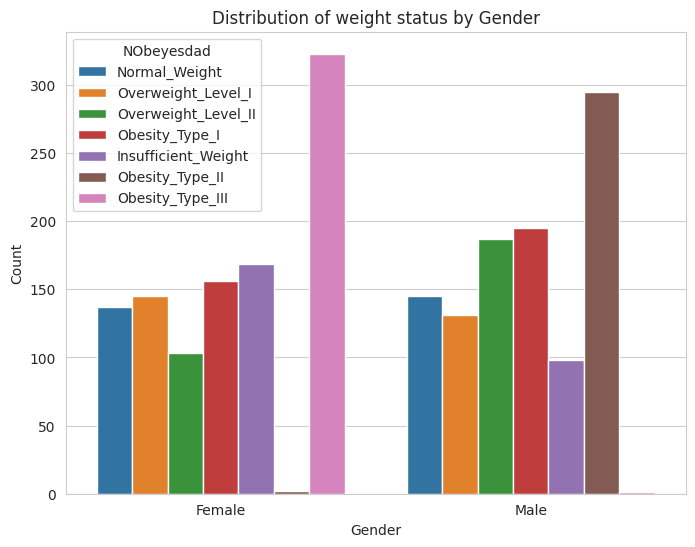
plt.title("Distribution of Weight Status by CAEC")

plt.xlabel("CAEC")

plt.ylabel("Count")

plt.legend(title="Weight Status")

plt.show()

A chart with different colored squares

Description automatically generatedA graph of weight status

Description automatically generatedA graph of weight status

Description automatically generatedA graph of weight status

Description automatically generatedA graph of weight status

Description automatically generatedA graph of weight loss

Description automatically generated

dfnum=df.select\_dtypes(exclude='object') # selecting only numerical features

target=df["NObeyesdad"]

plt.figure(figsize=(20, 6))

sns.boxplot(data=dfnum, hue=target, y="Age")

plt.title("Age vs weight status (box plot)")

plt.show()

plt.figure(figsize=(20, 6))

sns.boxplot(data=dfnum, hue=target, y="Height")

plt.title("Height vs weight status (box plot)")

plt.show()

plt.figure(figsize=(20, 6))

sns.boxplot(data=dfnum, hue=target, y="NCP")

plt.title("NCP vs weight status (box plot)")

plt.show()

plt.figure(figsize=(20, 6))

sns.boxplot(data=dfnum, hue=target, y="CH2O")

plt.title("CH2O vs weight status (box plot)")

plt.show()

plt.figure(figsize=(20, 6))

sns.boxplot(data=dfnum, hue=target, y="FAF")

plt.title("FAF vs weight status (box plot)")

plt.show()

plt.figure(figsize=(20, 6))

sns.boxplot(data=dfnum, hue=target, y="TUE")

plt.title("TUE vs weight status (box plot)")

plt.show()



A diagram with colorful squares and lines

Description automatically generated with medium confidenceA diagram of a graph

Description automatically generated with medium confidenceA diagram of a graph

Description automatically generatedA graph showing different colored squares

Description automatically generated

Then I started training models. First I train a hybrid model combining FNN and GRU models.

# Hybrid model approach combining FNN and GRU

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, confusion\_matrix

import tensorflow as tf

from tensorflow.keras.layers import Input, Dense, Dropout, GRU, concatenate, Reshape, Add

from tensorflow.keras.models import Model

# Data Preprocessing

# Encode the categorical target variable because model cannot train on categorical variables so they are needed to be encoded in numerical form

label\_encoder = LabelEncoder()

df['NObeyesdad'] = label\_encoder.fit\_transform(df['NObeyesdad'])

# Split data into features (X) and target variable (y)

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

y = df['NObeyesdad']

# Perform one-hot encoding for categorical variables because categorical features need encoding and one-hot is a good way because it encode in such a way that variables do not lose their original classes

X = pd.get\_dummies(X)

# Split data into training and testing sets (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardize features by making the mean = 0 and standard deviation = 1 so that all features have same scale. This will eliminate biasness in model by making influence of each feature equal.

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

y\_train = label\_encoder.fit\_transform(y\_train)

y\_test= label\_encoder.transform(y\_test)

# Find the indices of categorical columns

categorical\_indices = np.where(np.any(np.isnan(X\_train), axis=0))[0]

# Find the indices of numerical columns

numerical\_indices = np.where(np.all(np.isfinite(X\_train), axis=0))[0]

# Separate categorical and numerical features for training data

X\_train\_categorical = X\_train[:, categorical\_indices]

X\_train\_numerical = X\_train[:, numerical\_indices]

# Separate categorical and numerical features for testing data

X\_test\_categorical = X\_test[:, categorical\_indices]

X\_test\_numerical = X\_test[:, numerical\_indices]

# Define the GRU model for processing categorical features because it is well suited for sequential data but we do not have sequential data and from categorical and numerical types categorical is more suited for GRU

input\_categorical = Input(shape=(X\_train\_categorical.shape[1],))

reshape\_categorical = Reshape((1, X\_train\_categorical.shape[1]))(input\_categorical)

# Reshaping because GRU is built for sequential data and expects 3D data containing timestamps and batch\_size so we have to reshape our categorical data and needed to add more dimensions so it will work with GRU model architecture

gru\_output = GRU(32)(reshape\_categorical)

gru\_model = Model(inputs=input\_categorical, outputs=gru\_output)

# Define the FNN model for processing numerical features because FNN is best suited for tabular data type

input\_numerical = Input(shape=(X\_train\_numerical.shape[1],))

dense1\_numerical = Dense(64, activation='relu')(input\_numerical)

dense2\_numerical = Dense(64, activation='relu')(dense1\_numerical)

fnn\_model = Model(inputs=input\_numerical, outputs=dense2\_numerical)

# Combine the outputs of both models

combined\_output = concatenate([gru\_model.output, fnn\_model.output])

dense1\_combined = Dense(64, activation='relu')(combined\_output)

output = Dense(7, activation='softmax')(dense1\_combined)  # 7 output neurons because we have 7 classes of our target variable

# Define the hybrid model

hybrid\_model = Model(inputs=[input\_categorical, input\_numerical], outputs=output)

# Compile the hybrid model

hybrid\_model.compile(optimizer='adam',

                     loss='sparse\_categorical\_crossentropy',

                     metrics=['accuracy'])

# Train the hybrid model

hybrid\_model.fit([X\_train\_categorical, X\_train\_numerical], y\_train, epochs=50, batch\_size=32, validation\_split=0.1)

# Make predictions on test data

y\_pred = hybrid\_model.predict([X\_test\_categorical, X\_test\_numerical])

y\_pred\_classes = np.argmax(y\_pred, axis=1)

# Convert true labels to categorical

y\_test\_categorical = label\_encoder.inverse\_transform(y\_test)

# Calculate evaluation metrics

accuracy = accuracy\_score(y\_test\_categorical, y\_pred\_classes)

precision = precision\_score(y\_test\_categorical, y\_pred\_classes, average='macro')

recall = recall\_score(y\_test\_categorical, y\_pred\_classes, average='macro')

f1 = f1\_score(y\_test\_categorical, y\_pred\_classes, average='macro')

auc\_roc = roc\_auc\_score(y\_test\_categorical, y\_pred, average='macro', multi\_class='ovr')

conf\_matrix = confusion\_matrix(y\_test\_categorical, y\_pred\_classes)

# Print evaluation metrics

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1-score:", f1)

print("AUC-ROC:", auc\_roc)

print("Confusion Matrix:")

print(conf\_matrix)

# Stacked model approach stacking first output again as input to the same model

# Combine the outputs of both models

combined\_output = concatenate([gru\_model.output, fnn\_model.output])

dense1\_combined = Dense(64, activation='relu')(combined\_output)

dense2\_combined = Dense(32, activation='relu')(dense1\_combined)

# Define the second stage model for classification

input\_second\_stage = Input(shape=(dense2\_combined.shape[1],))

dense\_second\_stage = Dense(64, activation='relu')(input\_second\_stage)

output = Dense(7, activation='softmax')(dense\_second\_stage)

# Define the second stage model

second\_stage\_model = Model(inputs=input\_second\_stage, outputs=output)

# Use the output of the first stage as input to the second stage model

output\_second\_stage = second\_stage\_model(dense2\_combined)

# Define the overall stacked model

stacked\_model = Model(inputs=[input\_categorical, input\_numerical], outputs=output\_second\_stage)

# Compile the stacked model

stacked\_model.compile(optimizer='adam',

                      loss='sparse\_categorical\_crossentropy',

                      metrics=['accuracy'])

# Train the stacked model

stacked\_model.fit([X\_train\_categorical, X\_train\_numerical], y\_train, epochs=50, batch\_size=32, validation\_split=0.1)

# Make predictions on test data

y\_pred = hybrid\_model.predict([X\_test\_categorical, X\_test\_numerical])

y\_pred\_classes = np.argmax(y\_pred, axis=1)

# Convert true labels to categorical

y\_test\_categorical = label\_encoder.inverse\_transform(y\_test)

# Calculate evaluation metrics

accuracy = accuracy\_score(y\_test\_categorical, y\_pred\_classes)

precision = precision\_score(y\_test\_categorical, y\_pred\_classes, average='macro')

recall = recall\_score(y\_test\_categorical, y\_pred\_classes, average='macro')

f1 = f1\_score(y\_test\_categorical, y\_pred\_classes, average='macro')

auc\_roc = roc\_auc\_score(y\_test\_categorical, y\_pred, average='macro', multi\_class='ovr')

conf\_matrix = confusion\_matrix(y\_test\_categorical, y\_pred\_classes)

# Print evaluation metrics

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1-score:", f1)

print("AUC-ROC:", auc\_roc)

print("Confusion Matrix:")

print(conf\_matrix)

# Another hybrid model approach combining ResNet and FNN

# ResNet-like architecture for processing categorical features because ResNet is best suited for image processing but in our case it it better suited for categorical data instead of numerical features

dense1\_categorical = Dense(64, activation='relu')(input\_categorical)

dense2\_categorical = Dense(64, activation='relu')(dense1\_categorical)

residual\_categorical = Add()([dense2\_categorical, dense1\_categorical])  # Residual connection

# Concatenate the outputs of both models

concatenated = concatenate([dense2\_numerical, residual\_categorical])

# Additional fully connected layers for combined features

dense\_combined = Dense(64, activation='relu')(concatenated)

dense\_combined = Dense(32, activation='relu')(dense\_combined)

# Output layer

output = Dense(7, activation='softmax')(dense\_combined)  # Adjust output units for your task

# Define the hybrid model

hybrid\_model = Model(inputs=[input\_categorical, input\_numerical], outputs=output)

# Compile the model

hybrid\_model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Train the model

hybrid\_model.fit([X\_train\_categorical, X\_train\_numerical], y\_train, epochs=50, batch\_size=32, validation\_split=0.1)

# Make predictions on test data

y\_pred = hybrid\_model.predict([X\_test\_categorical, X\_test\_numerical])

y\_pred\_classes = np.argmax(y\_pred, axis=1)

# Convert true labels to categorical

y\_test\_categorical = label\_encoder.inverse\_transform(y\_test)

# Calculate evaluation metrics

accuracy = accuracy\_score(y\_test\_categorical, y\_pred\_classes)

precision = precision\_score(y\_test\_categorical, y\_pred\_classes, average='macro')

recall = recall\_score(y\_test\_categorical, y\_pred\_classes, average='macro')

f1 = f1\_score(y\_test\_categorical, y\_pred\_classes, average='macro')

auc\_roc = roc\_auc\_score(y\_test\_categorical, y\_pred, average='macro', multi\_class='ovr')

conf\_matrix = confusion\_matrix(y\_test\_categorical, y\_pred\_classes)

# Print evaluation metrics

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1-score:", f1)

print("AUC-ROC:", auc\_roc)

print("Confusion Matrix:")

print(conf\_matrix)

**Evaluation of different models:**

**Hybrid model of FNN and GRU**

Accuracy: 0.9408983451536643

Precision: 0.9398158641733483

Recall: 0.9379489915243928

F1-score: 0.9387139206726836

AUC-ROC: 0.9937819138044084

Confusion Matrix:

[[54 2 0 0 0 0 0]

[ 2 55 0 0 0 4 1]

[ 0 0 76 2 0 0 0]

[ 0 0 1 57 0 0 0]

[ 0 0 0 0 63 0 0]

[ 0 7 0 0 0 48 1]

[ 0 0 1 0 0 4 45]]

Stacked model of FNN and GRU

Accuracy: 0.9401913875598086

Precision: 0.9386667346183154

Recall: 0.9392594288051714

F1-score: 0.9384045015537392

AUC-ROC: 0.9933213065259833

Confusion Matrix:

[[54 5 0 0 0 0 0]

[ 3 54 0 0 0 4 0]

[ 0 0 68 1 0 0 1]

[ 0 0 1 62 0 0 1]

[ 0 0 0 0 60 0 0]

[ 0 3 1 0 0 47 4]

[ 0 0 1 0 0 0 48]]

Hybrid model of ResNet and FNN

Accuracy: 0.9401913875598086

Precision: 0.9386667346183154

Recall: 0.9392594288051714

F1-score: 0.9384045015537392

AUC-ROC: 0.9933213065259833

Confusion Matrix:

[[54 5 0 0 0 0 0]

[ 3 54 0 0 0 4 0]

[ 0 0 68 1 0 0 1]

[ 0 0 1 62 0 0 1]

[ 0 0 0 0 60 0 0]

[ 0 3 1 0 0 47 4]

[ 0 0 1 0 0 0 48]]

**Discussion:**

**Model Performance:** All models demonstrated high accuracy and robustness in predicting obesity levels based on diverse features. The hybrid approach effectively leveraged the strengths of both FNN and GRU architectures.

**Limitations:** Limited insights into feature importance, potential bias in data collection, and assumptions in model selection and architecture.

**Improvements**: Further exploration of feature engineering, ensemble methods, and deep learning techniques like attention mechanisms for better model interpretability and performance.

**Conclusion:**

In conclusion, the developed machine learning models offer promising predictive capabilities for estimating obesity levels in individuals from Mexico, Peru, and Colombia. These models can serve as valuable tools for healthcare practitioners and policymakers in addressing the rising obesity epidemic and promoting healthier lifestyles.