Multi-class_Classification

October 13, 2025

1 Multi-class Classification

Estimated time needed: **30** mins

In this lab, you will learn the different strategies of Multi-class classification and implement the same on a real-world dataset.

1.1 Objectives

After completing this lab you will be able to:

1. Understand the use of one-hot encoding for categorical variables.
2. Implement logistic regression for multi-class classification using **One-vs-All (OvA)** and **One-vs-One (OvO)** strategies.
3. Evaluate model performance using appropriate metrics.

1.2 Import Necessary Libraries

First, to ensure the availability of the required libraries, execute the cell below.

[1]:
```
!pip install numpy==2.2.0
!pip install pandas==2.2.3
!pip install scikit-learn==1.6.0
!pip install matplotlib==3.9.3
!pip install seaborn==0.13.2
```

```
Collecting numpy==2.2.0
  Downloading
numpy-2.2.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(62 kB)
Downloading
numpy-2.2.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (16.1 MB)
                         16.1/16.1 MB
170.8 MB/s eta 0:00:00
Installing collected packages: numpy
Successfully installed numpy-2.2.0
Collecting pandas==2.2.3
  Downloading
pandas-2.2.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(89 kB)
```

```
Requirement already satisfied: numpy>=1.26.0 in /opt/conda/lib/python3.12/site-
packages (from pandas==2.2.3) (2.2.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/opt/conda/lib/python3.12/site-packages (from pandas==2.2.3) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.12/site-
packages (from pandas==2.2.3) (2024.2)
Collecting tzdata>=2022.7 (from pandas==2.2.3)
  Downloading tzdata-2025.2-py2.py3-none-any.whl.metadata (1.4 kB)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-
packages (from python-dateutil>=2.8.2->pandas==2.2.3) (1.17.0)
Downloading
pandas-2.2.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.7
MB)
                         12.7/12.7 MB
134.1 MB/s eta 0:00:00
Downloading tzdata-2025.2-py2.py3-none-any.whl (347 kB)
Installing collected packages: tzdata, pandas
Successfully installed pandas-2.2.3 tzdata-2025.2
Collecting scikit-learn==1.6.0
  Downloading scikit_learn-1.6.0-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (18 kB)
Requirement already satisfied: numpy>=1.19.5 in /opt/conda/lib/python3.12/site-
packages (from scikit-learn==1.6.0) (2.2.0)
Collecting scipy>=1.6.0 (from scikit-learn==1.6.0)
  Downloading
scipy-1.16.2-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata
(62 kB)
Collecting joblib>=1.2.0 (from scikit-learn==1.6.0)
  Downloading joblib-1.5.2-py3-none-any.whl.metadata (5.6 kB)
Collecting threadpoolctl>=3.1.0 (from scikit-learn==1.6.0)
  Downloading threadpoolctl-3.6.0-py3-none-any.whl.metadata (13 kB)
Downloading
scikit_learn-1.6.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(13.1 MB)
                         13.1/13.1 MB
144.5 MB/s eta 0:00:00
Downloading joblib-1.5.2-py3-none-any.whl (308 kB)
Downloading
scipy-1.16.2-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (35.7
MB)
                         35.7/35.7 MB
231.0 MB/s eta 0:00:00
Downloading threadpoolctl-3.6.0-py3-none-any.whl (18 kB)
Installing collected packages: threadpoolctl, scipy, joblib, scikit-learn
Successfully installed joblib-1.5.2 scikit-learn-1.6.0 scipy-1.16.2
threadpoolctl-3.6.0
Collecting matplotlib==3.9.3
  Downloading matplotlib-3.9.3-cp312-cp312-
```

```
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (11 kB)
Collecting contourpy>=1.0.1 (from matplotlib==3.9.3)
  Downloading contourpy-1.3.3-cp312-cp312-
manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata (5.5 kB)
Collecting cycler>=0.10 (from matplotlib==3.9.3)
  Downloading cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib==3.9.3)
  Downloading fonttools-4.60.0-cp312-cp312-
manylinux1_x86_64.manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_5_x86_6
4.whl.metadata (111 kB)
Collecting kiwisolver>=1.3.1 (from matplotlib==3.9.3)
  Downloading kiwisolver-1.4.9-cp312-cp312-
manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (6.3 kB)
Requirement already satisfied: numpy>=1.23 in /opt/conda/lib/python3.12/site-
packages (from matplotlib==3.9.3) (2.2.0)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.12/site-packages (from matplotlib==3.9.3) (24.2)
Collecting pillow>=8 (from matplotlib==3.9.3)
  Downloading pillow-11.3.0-cp312-cp312-
manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata (9.0 kB)
Collecting pyparsing>=2.3.1 (from matplotlib==3.9.3)
  Downloading pyparsing-3.2.5-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/conda/lib/python3.12/site-packages (from matplotlib==3.9.3) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-
packages (from python-dateutil>=2.7->matplotlib==3.9.3) (1.17.0)
Downloading
matplotlib-3.9.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (8.3
MB)
                         8.3/8.3 MB
142.1 MB/s eta 0:00:00
Downloading
contourpy-1.3.3-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (362
kB)
Downloading cycler-0.12.1-py3-none-any.whl (8.3 kB)
Downloading fonttools-4.60.0-cp312-cp312-
manylinux1_x86_64.manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_5_x86_6
4.whl (4.9 MB)
                         4.9/4.9 MB
143.7 MB/s eta 0:00:00
Downloading
kiwisolver-1.4.9-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (1.5
MB)
                         1.5/1.5 MB
89.4 MB/s eta 0:00:00
Downloading
pillow-11.3.0-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (6.6
MB)
```

```
                         6.6/6.6 MB
100.7 MB/s eta 0:00:00
Downloading pyparsing-3.2.5-py3-none-any.whl (113 kB)
Installing collected packages: pyparsing, pillow, kiwisolver, fonttools, cycler,
contourpy, matplotlib
Successfully installed contourpy-1.3.3 cycler-0.12.1 fonttools-4.60.0
kiwisolver-1.4.9 matplotlib-3.9.3 pillow-11.3.0 pyparsing-3.2.5
Collecting seaborn==0.13.2
  Downloading seaborn-0.13.2-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in
/opt/conda/lib/python3.12/site-packages (from seaborn==0.13.2) (2.2.0)
Requirement already satisfied: pandas>=1.2 in /opt/conda/lib/python3.12/site-
packages (from seaborn==0.13.2) (2.2.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in
/opt/conda/lib/python3.12/site-packages (from seaborn==0.13.2) (3.9.3)
Requirement already satisfied: contourpy>=1.0.1 in
/opt/conda/lib/python3.12/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.12/site-
packages (from matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/conda/lib/python3.12/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (4.60.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
/opt/conda/lib/python3.12/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (1.4.9)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.12/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (24.2)
Requirement already satisfied: pillow>=8 in /opt/conda/lib/python3.12/site-
packages (from matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/opt/conda/lib/python3.12/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (3.2.5)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/conda/lib/python3.12/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.12/site-
packages (from pandas>=1.2->seaborn==0.13.2) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /opt/conda/lib/python3.12/site-
packages (from pandas>=1.2->seaborn==0.13.2) (2025.2)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-
packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn==0.13.2)
(1.17.0)
Downloading seaborn-0.13.2-py3-none-any.whl (294 kB)
Installing collected packages: seaborn
Successfully installed seaborn-0.13.2
```

Now, import the necessary libraries for data processing, model training, and evaluation.

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.multiclass import OneVsOneClassifier
from sklearn.metrics import accuracy_score

import warnings
warnings.filterwarnings('ignore')
```

1.3 About the dataset

The data set being used for this lab is the “Obesity Risk Prediction” data set publically available on UCI Library under the CCA 4.0 license. The data set has 17 attributes in total along with 2,111 samples.

The attributes of the dataset are descibed below.

Variable Name

Type

Description

Gender

Categorical

Age

Continuous

Height

Continuous

Weight

Continuous

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

1.3.1 Load the dataset

Load the data set by executing the code cell below.

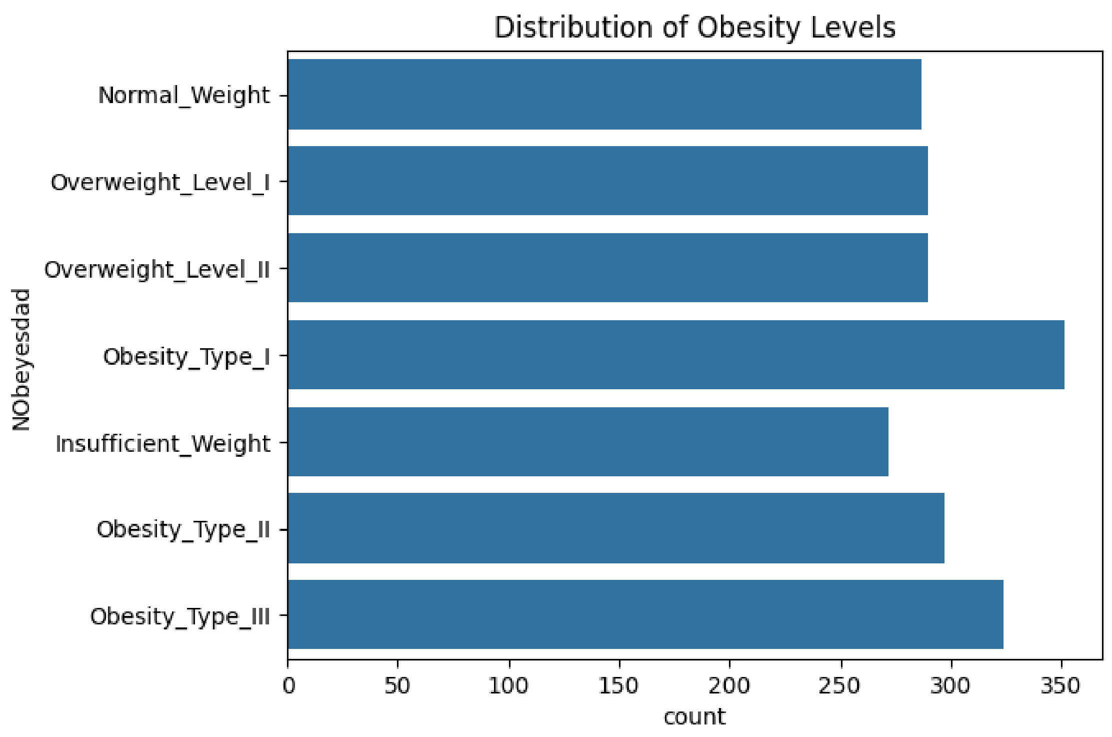
[3]:
```
file_path = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
  ↪GkDzb7bWrtvGXdPOfk6CIg/Obesity-level-prediction-dataset.csv"
data = pd.read_csv(file_path)
data.head()
```

[3]:
```
   Gender   Age  Height  Weight family_history_with_overweight FAVC  FCVC  \
0  Female  21.0    1.62    64.0                            yes   no   2.0
1  Female  21.0    1.52    56.0                            yes   no   3.0
2    Male  23.0    1.80    77.0                            yes   no   2.0
3    Male  27.0    1.80    87.0                             no   no   3.0
4    Male  22.0    1.78    89.8                             no   no   2.0

   NCP       CAEC SMOKE  CH2O  SCC  FAF  TUE        CALC  \
0  3.0  Sometimes    no   2.0   no  0.0  1.0          no
1  3.0  Sometimes   yes   3.0  yes  3.0  0.0   Sometimes
2  3.0  Sometimes    no   2.0   no  2.0  1.0  Frequently
3  3.0  Sometimes    no   2.0   no  2.0  0.0  Frequently
4  1.0  Sometimes    no   2.0   no  0.0  0.0   Sometimes

                  MTRANS           NObeyesdad
0  Public_Transportation        Normal_Weight
1  Public_Transportation        Normal_Weight
2  Public_Transportation        Normal_Weight
3                Walking   Overweight_Level_I
4  Public_Transportation  Overweight_Level_II
```

1.4 Exploratory Data Analysis

Visualize the distribution of the target variable to understand the class balance.

[4]:
```
# Distribution of target variable
sns.countplot(y='NObeyesdad', data=data)
plt.title('Distribution of Obesity Levels')
plt.show()
```



This shows that the dataset is fairly balanced and does not require any special attention in terms of biased training.

1.4.1 Exercise 1

Check for null values, and display a summary of the dataset (use `.info()` and `.describe()` methods).

[10]:
```
# your code here
#check Null values
print(data.isnull().sum())
print(data.info())
print(data.describe())
```

```
Gender                            0
Age                               0
Height                            0
Weight                            0
family_history_with_overweight    0
FAVC                              0
FCVC                              0
NCP                               0
CAEC                              0
SMOKE                             0
CH2O                              0
```

```
SCC                               0
FAF                               0
TUE                               0
CALC                              0
MTRANS                            0
NObeyesdad                        0
dtype: int64
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2111 entries, 0 to 2110
Data columns (total 17 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Gender                          2111 non-null   object
 1   Age                             2111 non-null   float64
 2   Height                          2111 non-null   float64
 3   Weight                          2111 non-null   float64
 4   family_history_with_overweight  2111 non-null   object
 5   FAVC                            2111 non-null   object
 6   FCVC                            2111 non-null   float64
 7   NCP                             2111 non-null   float64
 8   CAEC                            2111 non-null   object
 9   SMOKE                           2111 non-null   object
 10  CH2O                            2111 non-null   float64
 11  SCC                             2111 non-null   object
 12  FAF                             2111 non-null   float64
 13  TUE                             2111 non-null   float64
 14  CALC                            2111 non-null   object
 15  MTRANS                          2111 non-null   object
 16  NObeyesdad                      2111 non-null   object
dtypes: float64(8), object(9)
memory usage: 280.5+ KB
None
               Age       Height       Weight         FCVC          NCP  \
count  2111.000000  2111.000000  2111.000000  2111.000000  2111.000000
mean     24.312600     1.701677    86.586058     2.419043     2.685628
std       6.345968     0.093305    26.191172     0.533927     0.778039
min      14.000000     1.450000    39.000000     1.000000     1.000000
25%      19.947192     1.630000    65.473343     2.000000     2.658738
50%      22.777890     1.700499    83.000000     2.385502     3.000000
75%      26.000000     1.768464   107.430682     3.000000     3.000000
max      61.000000     1.980000   173.000000     3.000000     4.000000

              CH2O          FAF          TUE
count  2111.000000  2111.000000  2111.000000
mean      2.008011     1.010298     0.657866
std       0.612953     0.850592     0.608927
min       1.000000     0.000000     0.000000
25%       1.584812     0.124505     0.000000
```

```
50%       2.000000     1.000000     0.625350
75%       2.477420     1.666678     1.000000
max       3.000000     3.000000     2.000000
```

Click here for the solution

```
# Checking for null values
print(data.isnull().sum())

# Dataset summary
print(data.info())
print(data.describe())
```

Expected Output:

- Counts of null values for each column (likely zero for this dataset).
- Dataset info including column names, data types, and memory usage.
- Descriptive statistics for numerical columns.

1.5 Preprocessing the data

1.5.1 Feature scaling

Scale the numerical features to standardize their ranges for better model performance.

[14]:
```
# Standardizing continuous numerical features
continuous_columns = data.select_dtypes(include=['float64']).columns.tolist()

scaler = StandardScaler()
scaled_features = scaler.fit_transform(data[continuous_columns])

# Converting to a DataFrame
scaled_df = pd.DataFrame(scaled_features, columns=scaler.
 ↪get_feature_names_out(continuous_columns))

# Combining with the original dataset
scaled_data = pd.concat([data.drop(columns=continuous_columns), scaled_df],␣
 ↪axis=1)
data.sample(10)
```

[14]:
```
      Gender        Age    Height      Weight family_history_with_overweight  \
986   Female  28.583944  1.578560   65.522744                            yes
1357    Male  18.000000  1.784402  108.413119                            yes
647   Female  20.744839  1.667852   49.803921                            yes
782   Female  32.593129  1.721903   72.748903                            yes
166   Female  29.000000  1.740000   72.000000                            yes
303   Female  16.000000  1.570000   49.000000                             no
1703    Male  22.580038  1.849507  121.560938                            yes
157     Male  21.000000  1.670000   60.000000                            yes
232   Female  51.000000  1.590000   50.000000                            yes
```

```
2046  Female  25.895546  1.626179  110.074019                            yes

     FAVC      FCVC       NCP        CAEC SMOKE      CH2O  SCC       FAF  \
986    no  2.000000  2.283673   Sometimes    no  1.970622   no  1.863258
1357  yes  2.000000  2.475444   Sometimes    no  2.655795   no  1.000000
647   yes  2.977018  3.193671  Frequently    no  2.482933   no  2.000000
782   yes  2.000000  3.000000   Sometimes    no  1.000000   no  0.000000
166    no  3.000000  3.000000      Always    no  2.000000   no  0.000000
303   yes  2.000000  4.000000      Always    no  2.000000   no  0.000000
1703  yes  3.000000  2.272801   Sometimes    no  1.560064   no  0.796770
157   yes  2.000000  3.000000  Frequently    no  2.000000   no  2.000000
232    no  3.000000  3.000000   Sometimes   yes  3.000000  yes  2.000000
2046  yes  3.000000  3.000000   Sometimes    no  1.967707   no  0.014370

           TUE       CALC                 MTRANS           NObeyesdad
986   1.581703  Sometimes  Public_Transportation  Overweight_Level_II
1357  0.169879         no  Public_Transportation       Obesity_Type_I
647   1.000000         no  Public_Transportation  Insufficient_Weight
782   1.339232  Sometimes             Automobile   Overweight_Level_I
166   0.000000  Sometimes             Automobile        Normal_Weight
303   1.000000  Sometimes  Public_Transportation        Normal_Weight
1703  0.000000  Sometimes  Public_Transportation      Obesity_Type_II
157   1.000000  Sometimes  Public_Transportation        Normal_Weight
232   0.000000         no  Public_Transportation        Normal_Weight
2046  0.434073  Sometimes  Public_Transportation     Obesity_Type_III
```

Standardization of data is important to better define the decision boundaries between classes by making sure that the feature variations are in similar scales. The data is now ready to be used for training and testing.

1.5.2 One-hot encoding

Convert categorical variables into numerical format using one-hot encoding.

[15]:
```
# Identifying categorical columns
categorical_columns = scaled_data.select_dtypes(include=['object']).columns.
 ↪tolist()
categorical_columns.remove('NObeyesdad')  # Exclude target column

# Applying one-hot encoding
encoder = OneHotEncoder(sparse_output=False, drop='first')
encoded_features = encoder.fit_transform(scaled_data[categorical_columns])

# Converting to a DataFrame
encoded_df = pd.DataFrame(encoded_features, columns=encoder.
 ↪get_feature_names_out(categorical_columns))

# Combining with the original dataset
```

```
prepped_data = pd.concat([scaled_data.drop(columns=categorical_columns),␣
↪encoded_df], axis=1)
```

You will observe that all the categorical variables have now been modified to one-hot encoded features. This increases the overall number of fields to 24.

1.5.3 Encode the target variable

[16]:
```
# Encoding the target variable
prepped_data['NObeyesdad'] = prepped_data['NObeyesdad'].astype('category').cat.
↪codes
prepped_data.head()
```

[16]:
```
   NObeyesdad       Age    Height    Weight      FCVC       NCP      CH2O  \
0           1 -0.522124 -0.875589 -0.862558 -0.785019  0.404153 -0.013073
1           1 -0.522124 -1.947599 -1.168077  1.088342  0.404153  1.618759
2           1 -0.206889  1.054029 -0.366090 -0.785019  0.404153 -0.013073
3           5  0.423582  1.054029  0.015808  1.088342  0.404153 -0.013073
4           6 -0.364507  0.839627  0.122740 -0.785019 -2.167023 -0.013073

        FAF       TUE  Gender_Male  …  CAEC_no  SMOKE_yes  SCC_yes  \
0 -1.188039  0.561997          0.0  …      0.0        0.0      0.0
1  2.339750 -1.080625          0.0  …      0.0        1.0      1.0
2  1.163820  0.561997          1.0  …      0.0        0.0      0.0
3  1.163820 -1.080625          1.0  …      0.0        0.0      0.0
4 -1.188039 -1.080625          1.0  …      0.0        0.0      0.0

   CALC_Frequently  CALC_Sometimes  CALC_no  MTRANS_Bike  MTRANS_Motorbike  \
0              0.0             0.0      1.0          0.0               0.0
1              0.0             1.0      0.0          0.0               0.0
2              1.0             0.0      0.0          0.0               0.0
3              1.0             0.0      0.0          0.0               0.0
4              0.0             1.0      0.0          0.0               0.0

   MTRANS_Public_Transportation  MTRANS_Walking
0                           1.0             0.0
1                           1.0             0.0
2                           1.0             0.0
3                           0.0             1.0
4                           1.0             0.0

[5 rows x 24 columns]
```

1.5.4 Separate the input and target data

```
[17]: # Preparing final dataset
X = prepped_data.drop('NObeyesdad', axis=1)
y = prepped_data['NObeyesdad']
```

1.6 Model training and evaluation

1.6.1 Splitting the data set

Split the data into training and testing subsets.

```
[18]: # Splitting data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)
```

1.6.2 Logistic Regression with One-vs-All

In the One-vs-All approach:

- The algorithm trains a single binary classifier for each class.
- Each classifier learns to distinguish a single class from all the others combined.
- If there are k classes, k classifiers are trained.
- During prediction, the algorithm evaluates all classifiers on each input, and selects the class with the highest confidence score as the predicted class.

Advantages:

- Simpler and more efficient in terms of the number of classifiers (k)
- Easier to implement for algorithms that naturally provide confidence scores (e.g., logistic regression, SVM).

Disadvantages:

- Classifiers may struggle with class imbalance since each binary classifier must distinguish between one class and the rest.
- Requires the classifier to perform well even with highly imbalanced datasets, as the "all" group typically contains more samples than the "one" class.

Train a logistic regression model using the One-vs-All strategy and evaluate its performance.

```
[19]: # Training logistic regression model using One-vs-All (default)
model_ova = LogisticRegression(multi_class='ovr', max_iter=1000)
model_ova.fit(X_train, y_train)
```

```
[19]: LogisticRegression(max_iter=1000, multi_class='ovr')
```

You can now evaluate the accuracy of the trained model as a measure of its performance on unseen testing data.

[20]:
```
# Predictions
y_pred_ova = model_ova.predict(X_test)

# Evaluation metrics for OvA
print("One-vs-All (OvA) Strategy")
print(f"Accuracy: {np.round(100*accuracy_score(y_test, y_pred_ova),2)}%")
```

```
One-vs-All (OvA) Strategy
Accuracy: 76.12%
```

1.6.3 Logistic Regression with OvO

In the One-vs-One approach: * The algorithm trains a binary classifier for every pair of classes in the dataset. * If there are k classes, this results in $k(k-1)/2$ classifiers. * Each classifier is trained to distinguish between two specific classes, ignoring the rest. * During prediction, all classifiers are used, and a "voting" mechanism decides the final class by selecting the class that wins the majority of pairwise comparisons.

Advantages:

- Suitable for algorithms that are computationally expensive to train on many samples because each binary classifier deals with a smaller dataset (only samples from two classes).
- Can be more accurate in some cases since classifiers focus on distinguishing between two specific classes at a time.

Disadvantages:

- Computationally expensive for datasets with a large number of classes due to the large number of classifiers required.
- May lead to ambiguous predictions if voting results in a tie.

Train a logistic regression model using the One-vs-One (OvO) strategy and evaluate its performance.

[22]:
```
# Training logistic regression model using One-vs-One
model_ovo = OneVsOneClassifier(LogisticRegression(max_iter=1000))
model_ovo.fit(X_train, y_train)
```

[22]:
```
OneVsOneClassifier(estimator=LogisticRegression(max_iter=1000))
```

Evaluate the accuracy of the trained model as a measure of its performance on unseen testing data.

[24]:
```
# Predictions
y_pred_ovo = model_ovo.predict(X_test)

# Evaluation metrics for OvO
print("One-vs-One (OvO) Strategy")
print(f"Accuracy: {np.round(100*accuracy_score(y_test, y_pred_ovo),2)}%")
```

```
One-vs-One (OvO) Strategy
Accuracy: 92.2%
```

1.6.4 Exercises

Q1. Experiment with different test sizes in the train_test_split method (e.g., 0.1, 0.3) and observe the impact on model performance.

[30]:
```
# your code here
for test_size in [0.1,0.3]:
    X_train,X_test, y_train , y_test= train_test_split(X,y,␣
 ↪test_size=test_size, random_state=42, stratify=y)
    model_ovo.fit(X_train,y_train)
    y_pred = model_ova.predict(X_test)
    print(f'Test size: {test_size}')
    print("Accuracy:", accuracy_score(y_test, y_pred))
```

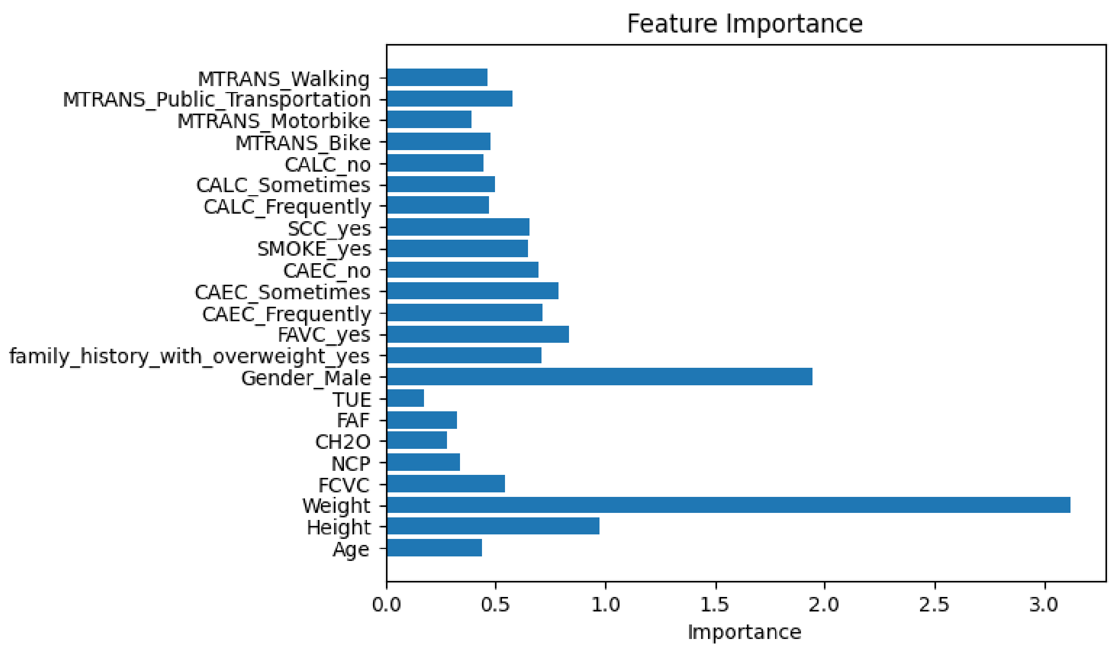
```
Test size: 0.1
Accuracy: 0.7641509433962265
Test size: 0.3
Accuracy: 0.7665615141955836
```

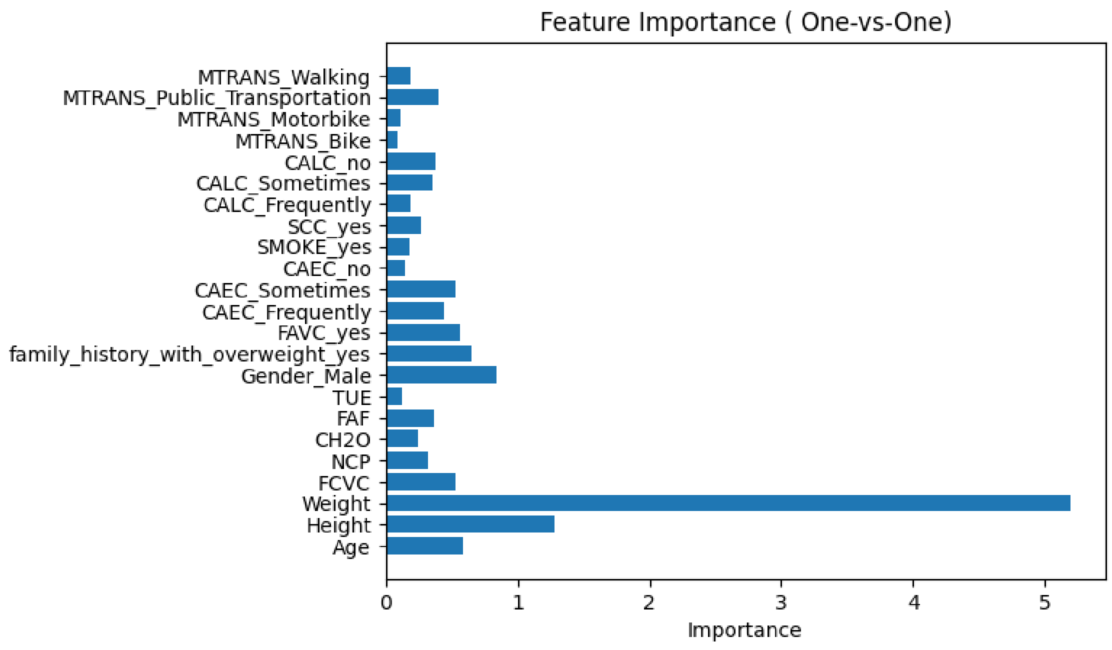
Click here for the solution

```
for test_size in [0.1, 0.3]:
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state
    model_ova.fit(X_train, y_train)
    y_pred = model_ova.predict(X_test)
    print(f"Test Size: {test_size}")
    print("Accuracy:", accuracy_score(y_test, y_pred))
```

Q2. Plot a bar chart of feature importance using the coefficients from the One vs All logistic regression model. Also try for the One vs One model.

[35]:
```
# your code here
#feature importance
feature_importance = np.mean(np.abs(model_ova.coef_), axis = 0)
plt.barh(X.columns, feature_importance)
plt.title("Feature Importance")
plt.xlabel("Importance")
plt.show()
# For One vs One model
# Collect all coefficients from each underlying binary classifier
coefs = np.array([est.coef_[0] for est in model_ovo.estimators_])

#NOw take the Mean accross all the those classifiers
feature_importance = np.mean(np.abs(coefs), axis=0)
 # plot feature importance
plt.barh(X.columns, feature_importance )
plt.title("Feature Importance ( One-vs-One)")
plt.xlabel("Importance")
plt.show()
```





Click here for the solution

```
# Feature importance
feature_importance = np.mean(np.abs(model_ova.coef_), axis=0)
plt.barh(X.columns, feature_importance)
plt.title("Feature Importance")
plt.xlabel("Importance")
plt.show()


# For One vs One model
# Collect all coefficients from each underlying binary classifier
coefs = np.array([est.coef_[0] for est in model_ovo.estimators_])

# Now take the mean across all those classifiers
feature_importance = np.mean(np.abs(coefs), axis=0)

# Plot feature importance
plt.barh(X.columns, feature_importance)
plt.title("Feature Importance (One-vs-One)")
plt.xlabel("Importance")
plt.show()
```

Q3. Write a function `obesity_risk_pipeline` to automate the entire pipeline:

Loading and preprocessing the data

Training the model

Evaluating the model

The function should accept the file path and test set size as the input arguments.

[37]:
```
# write your function here and then execute this cell
def obesity_risk_pipeline(data_path, test_size=0.2):.
# your code here
 # load data set
   data = pd.read_csv(data_path)
 # standardising continuous numerical features
continuous_columns= data.select_dtypes(include=["float64"].columns.tolist()
scalar = standardscalar()
scalar_feature= scalar.fit_transform(data[continuous_columns])


obesity_risk_pipeline(file_path, test_size=0.2)
```

```
  Cell In[37], line 2
    def obesity_risk_pipeline(data_path, test_size=0.2):.
                                                        ^
SyntaxError: invalid syntax
```

Click here for the solution

```
def obesity_risk_pipeline(data_path, test_size=0.2):
    # Load data
    data = pd.read_csv(data_path)

    # Standardizing continuous numerical features
    continuous_columns = data.select_dtypes(include=['float64']).columns.tolist()
    scaler = StandardScaler()
    scaled_features = scaler.fit_transform(data[continuous_columns])

    # Converting to a DataFrame
    scaled_df = pd.DataFrame(scaled_features, columns=scaler.get_feature_names_out(continuous_c

    # Combining with the original dataset
    scaled_data = pd.concat([data.drop(columns=continuous_columns), scaled_df], axis=1)

    # Identifying categorical columns
    categorical_columns = scaled_data.select_dtypes(include=['object']).columns.tolist()
    categorical_columns.remove('NObeyesdad')  # Exclude target column

    # Applying one-hot encoding
    encoder = OneHotEncoder(sparse_output=False, drop='first')
    encoded_features = encoder.fit_transform(scaled_data[categorical_columns])

    # Converting to a DataFrame
    encoded_df = pd.DataFrame(encoded_features, columns=encoder.get_feature_names_out(categori

    # Combining with the original dataset
    prepped_data = pd.concat([scaled_data.drop(columns=categorical_columns), encoded_df], axis

    # Encoding the target variable
    prepped_data['NObeyesdad'] = prepped_data['NObeyesdad'].astype('category').cat.codes

    # Preparing final dataset
    X = prepped_data.drop('NObeyesdad', axis=1)
    y = prepped_data['NObeyesdad']

    # Splitting data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_stat

    # Training and evaluation
    model = LogisticRegression(multi_class='multinomial', max_iter=1000)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print("Accuracy:", accuracy_score(y_test, y_pred))

# Call the pipeline function with file_path
obesity_risk_pipeline(file_path, test_size=0.2)
```

1.6.5 Congratulations! You're ready to move on to your next lesson!

1.7 Author

Abishek Gagneja

Other Contributors

Jeff Grossman

<!– ## Changelog

Date | Version | Changed by | Change Description |

|:————|:——|:—————|:——————————————|

2024-11-05 | 1.0 Abhishek Gagnejan | Fresh version created |
2025-05-13 | 1.1 Anita Verma | Added the solution code for Ovo model Q2 |