Multi-class_Classification

May 21, 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.

[ ]:
```
!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
```

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

[ ]:
```
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.

[ ]:
```
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()
```

1.4 Exploratory Data Analysis

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

[ ]:
```
# 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).

[ ]:
```
# your code here

print(data.info())
print(data.describe())
```

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.

[ ]:
```
# 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)
```

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.

[ ]:
```
# 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

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

1.5.4 Separate the input and target data

[ ]:
```
# 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.

```
[ ]: # 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.

```
[ ]: # 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)
```

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

```
[ ]: # 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)}%")
```

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.

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

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

```
[ ]: # 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)}%")
```

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.

```
[ ]: # your code here
```

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_stat
    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.

```
[ ]: # your code here
```

Click here for the solution

```
# Feature importance
feature_importance = np.mean(np.abs(model_ova.coef_), axis=0)
```

```python
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.

[ ]:
```python
# write your function here and then execute this cell
def obesity_risk_pipeline(data_path, test_size=0.2):
# your code here

obesity_risk_pipeline(file_path, test_size=0.2)
```

Click here for the solution

```python
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_

    # 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 |