CSC-722: Machine Learning Fundamentals

Assignemnt: Code for Logistic Regression & Naive Bayes

Submitted by: Prajwol Tiwari

Importing NumPy and Pandas libraries for numerical computations and data manipulation, respectively.

In [1]:
```python
import numpy as np
import pandas as pd
```

Installing scikit-learn package using pip, a Python package manager, to make it available for use in the current environment.

In [2]:
```python
!pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\prajw\anaconda3\lib\site-packages (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in c:\users\prajw\anaconda3\lib\site-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.3.2 in c:\users\prajw\anaconda3\lib\site-packages (from scikit-learn) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in c:\users\prajw\anaconda3\lib\site-packages (from scikit-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\prajw\anaconda3\lib\site-packages (from scikit-learn) (2.2.0)
```

Importing specific functions and classes from scikit-learn metrics module to use them for evaluating model performance.

In [3]:
```python
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, ConfusionMatrixDisplay,
    roc_curve, RocCurveDisplay, roc_auc_score, log_loss
)
```

In [4]:
```python
# Importing Matplotlib library for data visualization, particularly for creating plots.
import matplotlib.pyplot as plt

# Importing Seaborn library for statistical data visualization, providing a high-level interface for drawing attractive and informative
import seaborn as sns
```

Assignment:

Use this dataset:

https://archive.ics.uci.edu/dataset/244/fertility

fix and fit the code and Report the results

Installing the ucimlrepo package using pip to access datasets from the UCI Machine Learning Repository.

In [5]:
```python
!pip install ucimlrepo
```

```
Collecting ucimlrepo
  Downloading ucimlrepo-0.0.6-py3-none-any.whl.metadata (5.3 kB)
Downloading ucimlrepo-0.0.6-py3-none-any.whl (8.0 kB)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.6
```

In [6]:
```python
# Importing fetch_ucirepo function from ucimlrepo to fetch datasets from the UCI Machine Learning Repository.
from ucimlrepo import fetch_ucirepo

# Fetching the dataset with ID 244 from the UCI Machine Learning Repository.
fertility = fetch_ucirepo(id=244)

# Extracting the features and targets data from the fetched dataset and storing them in pandas dataframes.
X_df = fertility.data.features  # Features dataframe
y_df = fertility.data.targets   # Targets dataframe

# Printing the metadata information of the fetched dataset.
print(fertility.metadata)

# Printing the variable information of the fetched dataset.
print(fertility.variables)
```

```
{'uci_id': 244, 'name': 'Fertility', 'repository_url': 'https://archive.ics.uci.edu/dataset/244/fertility', 'data_url': 'https://archiv
e.ics.uci.edu/static/public/244/data.csv', 'abstract': '100 volunteers provide a semen sample analyzed according to the WHO 2010 criteri
a. Sperm concentration are related to socio-demographic data, environmental factors, health status, and life habits', 'area': 'Health an
d Medicine', 'tasks': ['Classification', 'Regression'], 'characteristics': ['Multivariate'], 'num_instances': 100, 'num_features': 9, 'f
eature_types': ['Real'], 'demographics': ['Age'], 'target_col': ['diagnosis'], 'index_col': None, 'has_missing_values': 'no', 'missing_v
alues_symbol': None, 'year_of_dataset_creation': 2012, 'last_updated': 'Fri Mar 15 2024', 'dataset_doi': '10.24432/C5Z01Z', 'creators':
['David Gil', 'Jose Girela'], 'intro_paper': {'title': 'Predicting seminal quality with artificial intelligence methods', 'authors': 'Da
vid Gil, J. L. Girela, Joaquin De Juan, M. Jose Gomez-Torres, Magnus Johnsson', 'published_in': 'Expert systems with applications', 'yea
r': 2012, 'url': 'https://www.semanticscholar.org/paper/Predicting-seminal-quality-with-artificial-methods-Gil-Girela/92759c5ee08b9e6e7b
17d1ccd48a7f8c02aba893', 'doi': None}, 'additional_info': {'summary': None, 'purpose': None, 'funded_by': None, 'instances_represent': N
one, 'recommended_data_splits': None, 'sensitive_data': None, 'preprocessing_description': None, 'variable_info': 'Season in which the a
nalysis was performed. \t1) winter, 2) spring, 3) Summer, 4) fall. \t(-1, -0.33, 0.33, 1) \r\n\r\nAge at the time of analysis. \t18-36
\t(0, 1) \r\n\r\nChildish diseases (ie , chicken pox, measles, mumps, polio)\t1) yes, 2) no. \t(0, 1) \r\n\r\nAccident or serious trauma
\t1) yes, 2) no. \t(0, 1) \r\n\r\nSurgical intervention \t1) yes, 2) no. \t(0, 1) \r\n\r\nHigh fevers in the last year \t1) less than th
ree months ago, 2) more than three months ago, 3) no. \t(-1, 0, 1) \r\n\r\nFrequency of alcohol consumption \t1) several times a day, 2)
every day, 3) several times a week, 4) once a week, 5) hardly ever or never \t(0, 1) \r\n\r\nSmoking habit \t1) never, 2) occasional 3)
daily. \t(-1, 0, 1) \r\n\r\nNumber of hours spent sitting per day \tene-16\t(0, 1) \r\n\r\nOutput: Diagnosis\tnormal (N), altered (O)\t
\r\n', 'citation': None}}
                    name     role         type demographic description units  \
0                 season  Feature   Continuous        None        None  None
1                    age  Feature      Integer         Age        None  None
2         child_diseases  Feature       Binary        None        None  None
3               accident  Feature       Binary        None        None  None
4  surgical_intervention  Feature       Binary        None        None  None
5            high_fevers  Feature  Categorical        None        None  None
6                alcohol  Feature  Categorical        None        None  None
7                smoking  Feature  Categorical        None        None  None
8            hrs_sitting  Feature      Integer        None        None  None
9              diagnosis   Target       Binary        None        None  None

  missing_values
0             no
1             no
2             no
3             no
4             no
5             no
6             no
7             no
8             no
9             no
```

In [7]:
```python
# Displaying the first few rows of the features dataframe (X_df) to inspect the structure and content of the data.
X_df.head()
```

Out[7]:

	season	age	child_diseases	accident	surgical_intervention	high_fevers	alcohol	smoking	hrs_sitting
0	-0.33	0.69	0	1	1	0	0.8	0	0.88
1	-0.33	0.94	1	0	1	0	0.8	1	0.31
2	-0.33	0.50	1	0	0	0	1.0	-1	0.50
3	-0.33	0.75	0	1	1	0	1.0	-1	0.38
4	-0.33	0.67	1	1	0	0	0.8	-1	0.50

In [8]:
```python
# Displaying the first few rows of the targets dataframe (y_df) to inspect the structure and content of the target variable data.
y_df.head()
```

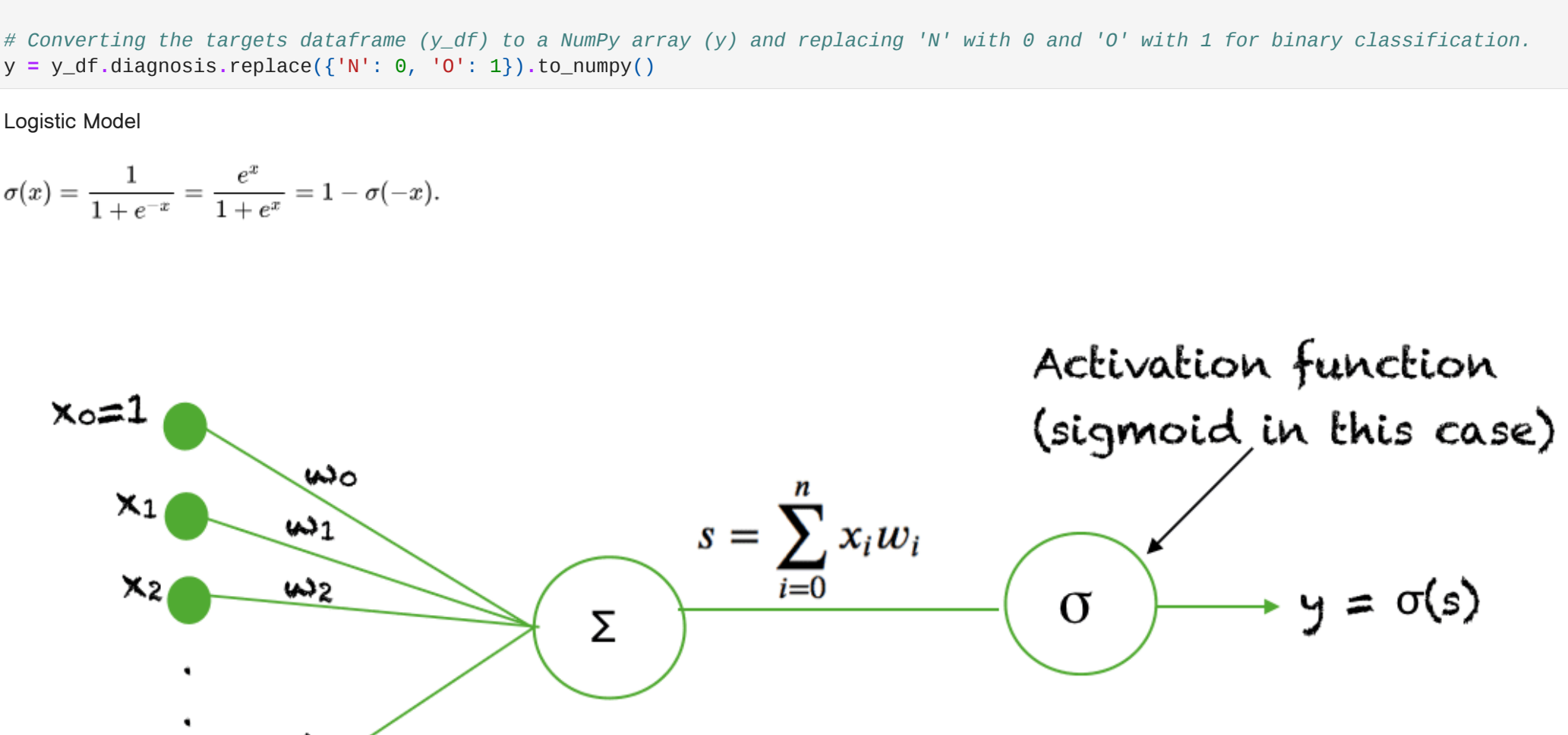
Out[8]:

	diagnosis
0	N
1	O
2	N
3	N
4	O

In [9]:
```python
# Converting the features dataframe (X_df) to a NumPy array (X) to prepare for implememnting Logistic Regression.
X = X_df.to_numpy()

# Converting the targets dataframe (y_df) to a NumPy array (y) and replacing 'N' with 0 and 'O' with 1 for binary classification.
y = y_df.diagnosis.replace({'N': 0, 'O': 1}).to_numpy()
```

Logistic Model

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x} = 1 - \sigma(-x).$$



Why The Sigmoid Function Is Important In Neural Networks?

If we use a linear activation function in a neural network, then this model can only learn linearly separable problems. However, with the addition of just one hidden layer and a sigmoid activation function in the hidden layer, the neural network can easily learn a non-linearly separable problem. Using a non-linear function produces non-linear boundaries and hence, the sigmoid function can be used in neural networks for learning complex decision functions.

The only non-linear function that can be used as an activation function in a neural network is one which is monotonically increasing. So for example, sin(x) or cos(x) cannot be used as activation functions. Also, the activation function should be defined everywhere and should be continuous everywhere in the space of real numbers. The function is also required to be differentiable over the entire space of real numbers.

Typically a back propagation algorithm uses gradient descent to learn the weights of a neural network. To derive this algorithm, the derivative of the activation function is required.

The fact that the sigmoid function is monotonic, continuous and differentiable everywhere, coupled with the property that its derivative can be expressed in terms of itself, makes it easy to derive the update equations for learning the weights in a neural network when using back propagation algorithm.

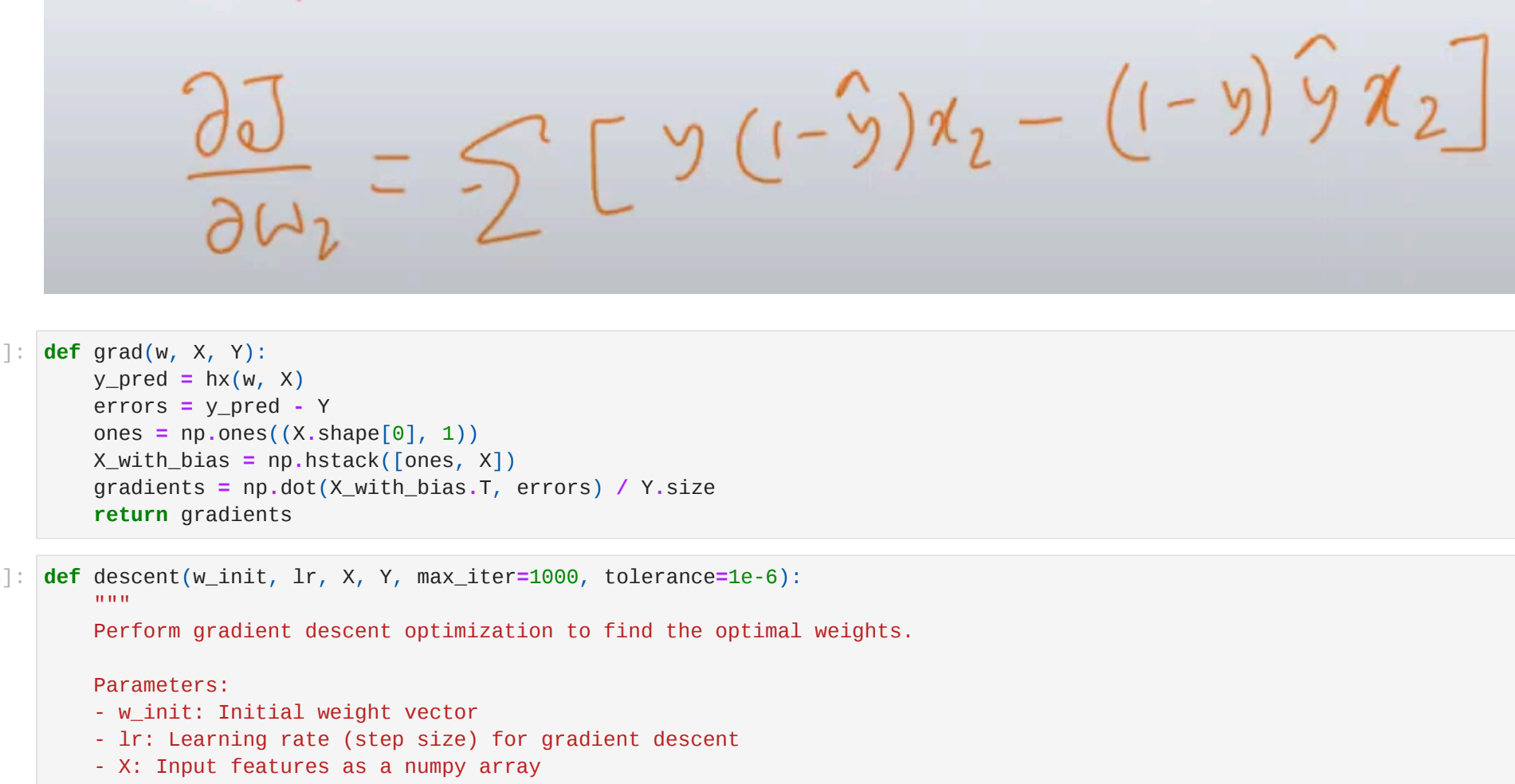
In [10]:
```python
# This function given below implements the sigmoid function, which is commonly used in logistic regression
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

In [11]:
```python
# This function calculates the hypothesis (hx) for logistic regression.
def hx(w,X):
    ones = np.ones((X.shape[0], 1))
    X_with_bias = np.hstack([ones, X])
    z = np.dot(X_with_bias, w)
    return sigmoid(z)
```

Cost Function - Binary Cross Entropy

In [12]:
```python
def cost(w, X, Y):
    y_pred = hx(w,X)
    return -np.mean(Y * np.log(y_pred) + (1 - Y) * np.log(1 - y_pred))
```

Gradient Descent



In [13]:
```python
def grad(w, X, Y):
    y_pred = hx(w, X)
    errors = y_pred - Y
    ones = np.ones((X.shape[0], 1))
    X_with_bias = np.hstack([ones, X])
    gradients = np.dot(X_with_bias.T, errors) / Y.size
    return gradients
```

In [14]:
```python
def descent(w_init, lr, X, Y, max_iter=1000, tolerance=1e-6):
    """
    Perform gradient descent optimization to find the optimal weights.

    Parameters:
    - w_init: Initial weight vector
    - lr: Learning rate (step size) for gradient descent
    - X: Input features as a numpy array
    - Y: Target labels as a numpy array
    - max_iter: Maximum number of iterations for gradient descent (default: 1000)
    - tolerance: Convergence tolerance for stopping criteria (default: 1e-6)

    Returns:
    - Optimized weight vector
    """
    w = w_init
    for j in range(max_iter):
        gradients = grad(w, X, Y)
        w_new = w - lr * gradients
        if np.linalg.norm(w_new - w, 2) < tolerance:
            print(f"Converged after {j + 1} iterations.")
            return w_new
        w = w_new
        if (j + 1) % 100 == 0:
            print(f"Iteration {j + 1}: Cost {cost(w, X, Y)}")
            print(f"Weights: {w}")
    print("Max iterations reached without convergence.")
    return w
```

Presenting Results Function:

Initializing Parameters

In [15]:
```python
def visualize_evaluation_metrics(y_test, y_pred, y_proba=None):
    """
    Compute evaluation metrics and display visualizations.

    Parameters:
    - y_test: True labels of the test set
    - y_pred: Predicted labels of the test set
    - y_proba: Predicted probabilities

    """

    # Compute evaluation metrics
    metrics = {
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred),
        "F1 Score": f1_score(y_test, y_pred),
        "AUC": roc_auc_score(y_test, y_pred),
        "Log loss": log_loss(y_test, y_pred),
    }

    # Print evaluation metrics
    print("Evaluation Metrics:")
    for metric, value in metrics.items():
        print(f"{metric}: {value}")
    print()

    # Display Confusion Matrix plot
    cm = confusion_matrix(y_test, y_pred)
    disp_cm = ConfusionMatrixDisplay(confusion_matrix=cm)
    disp_cm.plot()
    plt.title('Confusion Matrix')
    plt.show()

    # Display ROC Curve plot
    fpr, tpr, _ = roc_curve(y_test, y_proba if y_proba is not None else y_pred)
    roc_auc = roc_auc_score(y_test, y_proba if y_proba is not None else roc_auc_score(y_test, y_pred)
    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, color='blue', label='ROC Curve (AUC = {:.2f})'.format(roc_auc))
    plt.plot([0, 1], [0, 1], color='green', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc='lower right')
    plt.grid(True)
    plt.show()
```

In [16]:
```python
# Initializing the weight vector w_init with zeros, adding one additional element for the bias term.
w_init = np.zeros(X.shape[1] + 1)

# Setting the learning rate lr to 0.01 for gradient descent optimization.
lr = 0.01
```

Training the Model

In [17]:
```python
# Performing gradient descent optimization to find the optimal weights
w_optimal = descent(w_init, lr, X, y, 1000, 1e-6)

# Print the optimal weights obtained after training the model.
print(f"\nOptimal weights after training Logistic Regression model:")
print(w_optimal)
```

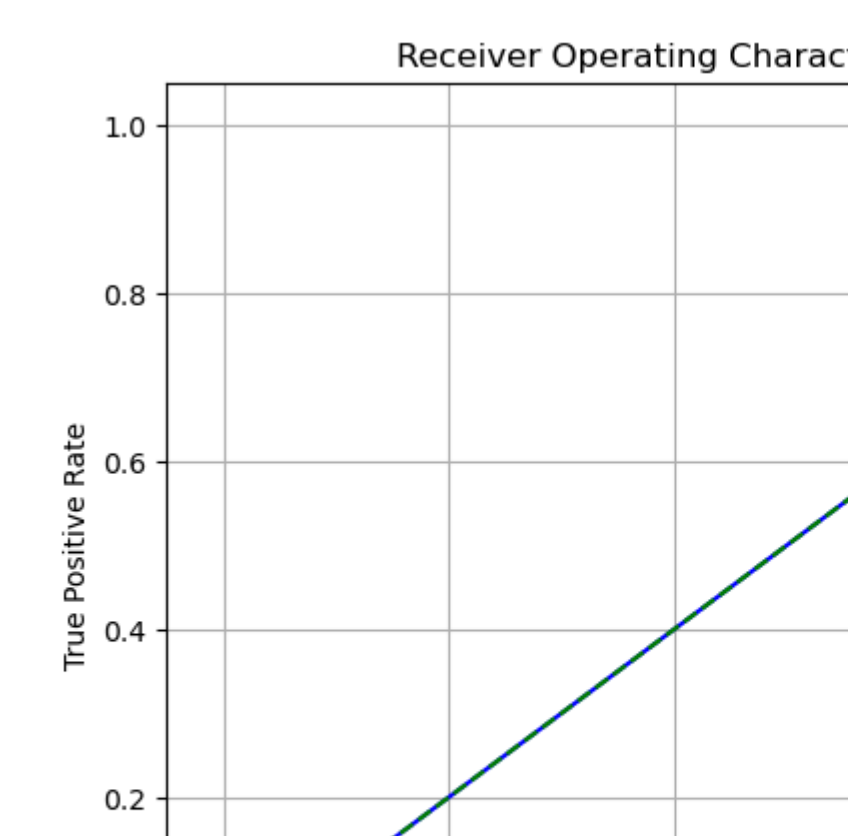
```
Iteration 100: Cost 0.4354144569747552
Weights: [-0.25213367  0.06214325 -0.16364031 -0.22027804 -0.12934375 -0.11664533
 -0.06819453 -0.21693991  0.09301912 -0.10129138]
Iteration 200: Cost 0.3798059970753206
Weights: [-0.36956183  0.10632317 -0.23679483 -0.32026913 -0.19576786 -0.16247225
 -0.10838357 -0.32125459  0.13358173 -0.147873  ]
Iteration 300: Cost 0.36189020717787557
Weights: [-0.4342601   0.14340965 -0.27464901 -0.37354977 -0.23816295 -0.18143329
 -0.13765424 -0.38159125  0.15516677 -0.17301437]
Iteration 400: Cost 0.35432965341921785
Weights: [-0.47340047  0.17654133 -0.29543329 -0.40425669 -0.2690011  -0.18772299
 -0.16146321 -0.42064174  0.16806627 -0.18774766]
Iteration 500: Cost 0.3503795840111965
Weights: [-0.4984162   0.20688017 -0.30680088 -0.42246229 -0.29340023 -0.1872079
 -0.18203867 -0.44793453  0.1763724  -0.19671612]
Iteration 600: Cost 0.3479050028352468
Weights: [-0.51500724  0.23495813 -0.31256145 -0.43316069 -0.31386821 -0.18277927
 -0.20047005 -0.46821317  0.18204551 -0.20223478]
Iteration 700: Cost 0.34611759821097265
Weights: [-0.5263383   0.26106513 -0.31482795 -0.43911656 -0.33178187 -0.17602981
 -0.21735719 -0.48410883  0.18614235 -0.20558894]
Iteration 800: Cost 0.34469387389397965
Weights: [-0.53429279  0.28538639 -0.31485683 -0.44197446 -0.34795561 -0.16790146
 -0.233059   -0.4971863   0.18927683 -0.2075434 ]
Iteration 900: Cost 0.3434893062673253
Weights: [-0.54004344  0.30805829 -0.31342988 -0.4427606  -0.36289615 -0.15897698
 -0.24780413 -0.50841954  0.19182272 -0.20857438]
Iteration 1000: Cost 0.34243374013507155
Weights: [-0.54434237  0.32919307 -0.31104814 -0.44213641 -0.37693011 -0.14962702
 -0.26174597 -0.51843299  0.19401388 -0.20898728]
Max iterations reached without convergence.

Optimal weights after training Logistic Regression model:
[-0.54434237  0.32919307 -0.31104814 -0.44213641 -0.37693011 -0.14962702
 -0.26174597 -0.51843299  0.19401388 -0.20898728]
```

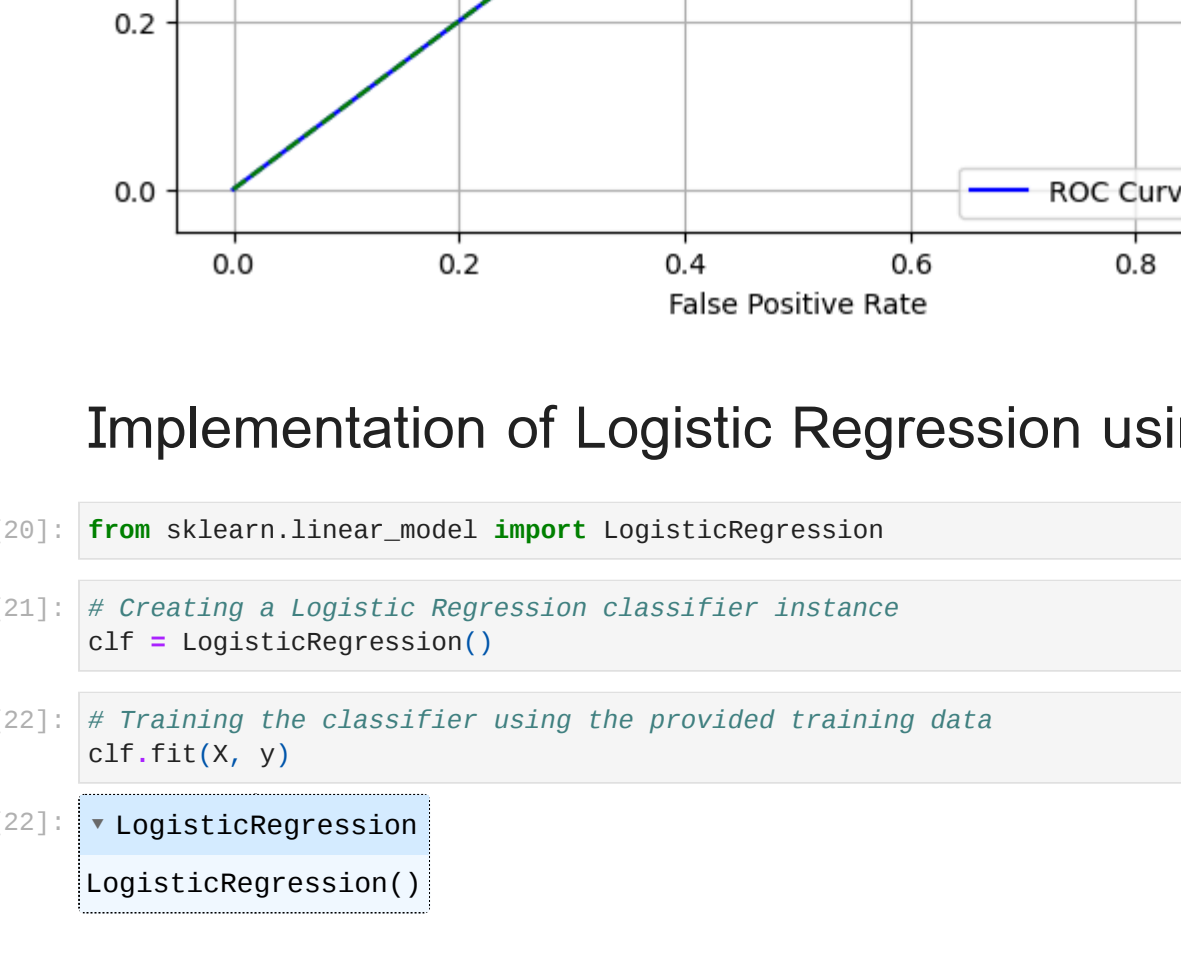
In [18]:
```python
# Computing the predicted probabilities using the trained model and optimal weights.
y_prob = hx(w_optimal, X)

# Converting the predicted probabilities to binary predictions based on a threshold of 0.5.
y_pred = np.array([1 if prob > 0.5 else 0 for prob in y_prob])
```

In [19]:
```python
# Generating and displaying evaluation metrics and visualizations for the model's predictions.
visualize_evaluation_metrics(y, y_pred)
```

```
Evaluation Metrics:
Accuracy: 0.88
Precision: 0.0
Recall: 0.0
F1 Score: 0.0
AUC: 0.5
Log loss: 4.325238406694059
```

```
C:\Users\prajw\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```





Implementation of Logistic Regression using the Scikit-learn library

In [20]:
```python
from sklearn.linear_model import LogisticRegression
```

In [21]:
```python
# Creating a Logistic Regression classifier instance
clf = LogisticRegression()
```

In [22]:
```python
# Training the classifier using the provided training data
clf.fit(X, y)
```

Out[22]:

```
▾ LogisticRegression
LogisticRegression()
```

In [23]:
```python
# Calculating and displaying the accuracy score of the classifier
accuracy = clf.score(X, y)
print(f"Accuracy: {accuracy}")
```

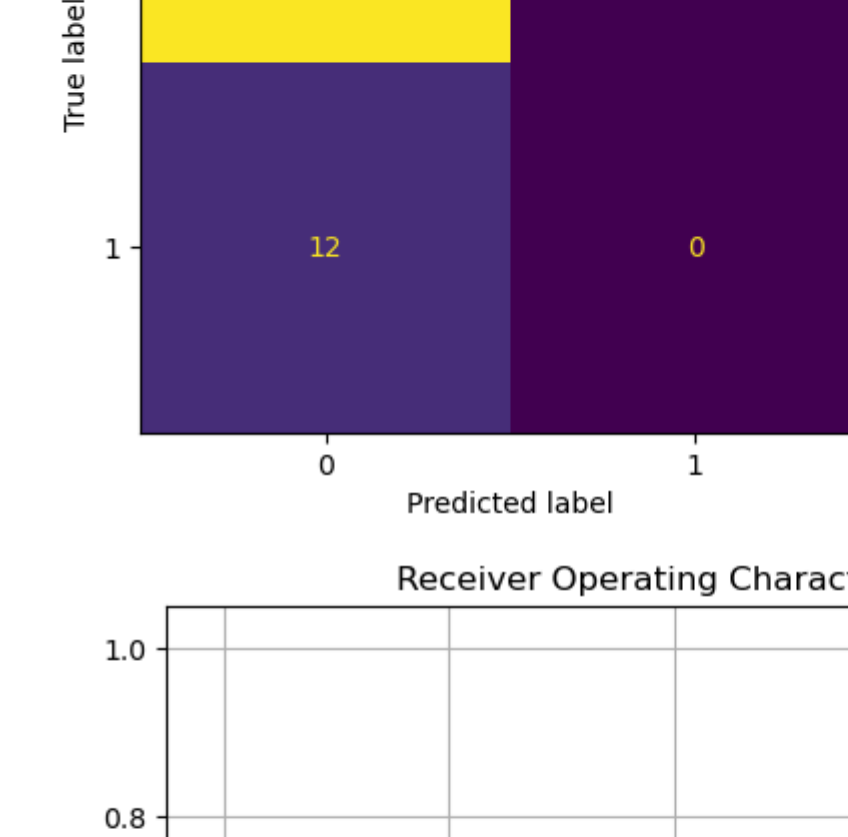
```
Accuracy: 0.88
```

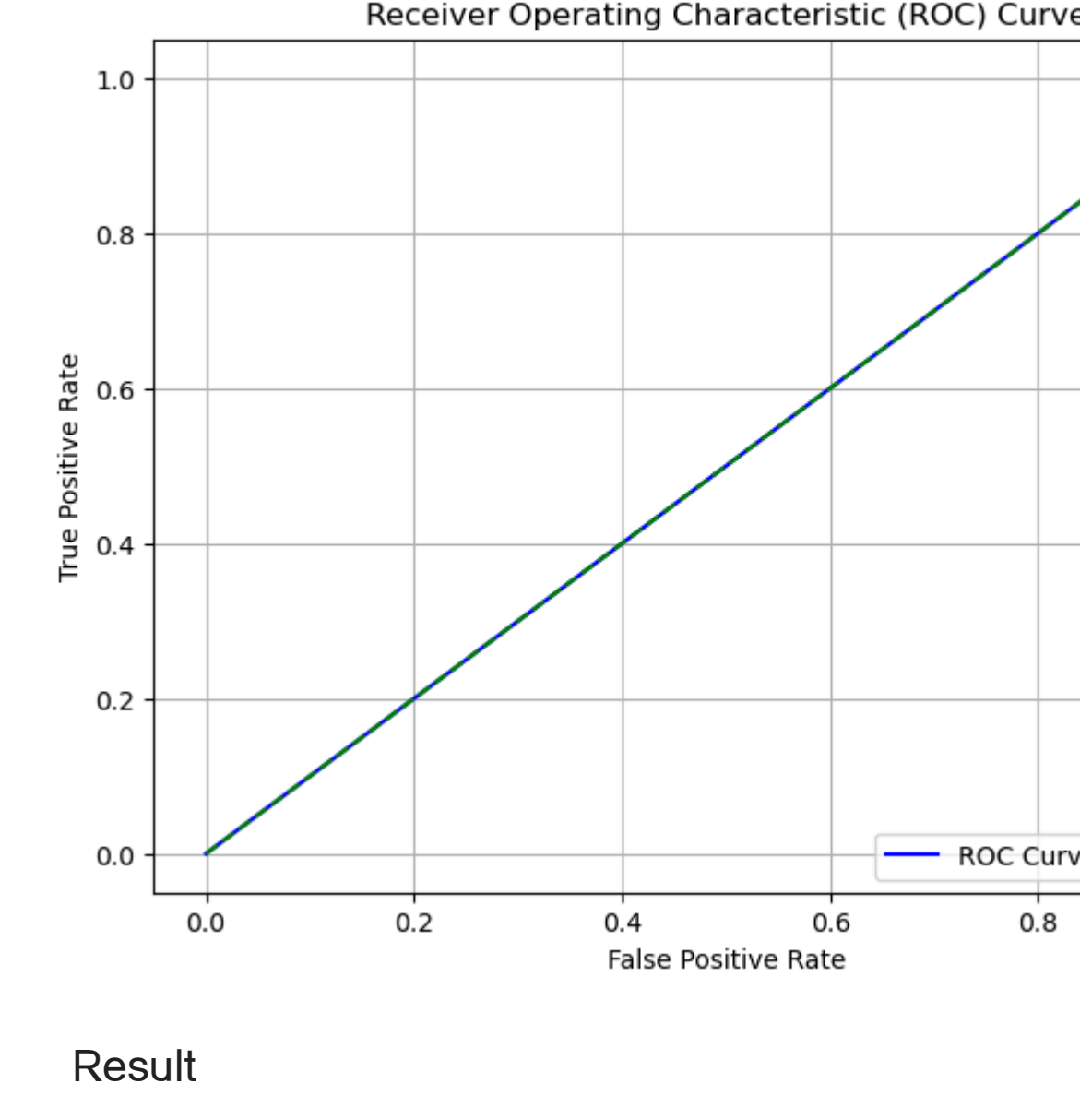
In [24]:
```python
# Generating predictions and probabilities using the trained classifier
y_prob_clf = clf.predict_proba(X)
y_pred_clf = clf.predict(X)
```

In [25]:
```python
# Calling the generate_results function to evaluate the classifier's predictions
visualize_evaluation_metrics(y, y_pred_clf)
```

```
Evaluation Metrics:
Accuracy: 0.88
Precision: 0.0
Recall: 0.0
F1 Score: 0.0
AUC: 0.5
Log loss: 4.325238406694059
```

```
C:\Users\prajw\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```





Result

The results obtained from both the Scratch Logistic Regression implementation and the Logistic Regression implementation using Scikit-Learn are identical and consistent. This consistency indicates that the Scratch implementation accurately replicates the behavior of the Scikit-Learn library methods.

However, an UndefinedMetricWarning is raised, suggesting that the model made no positive predictions, resulting in an ill-defined precision metric. This issue arises because both implementations struggle to predict the 'O' or 1 class, especially in the context of an imbalanced dataset where the 'O' target class is underrepresented, leading to difficulties in learning its patterns.