Cancer detection using logistic regression.

Made By: Ramesh Chandra Soren
Enrollment No: 2022CSB086

Logistic Regression

- Logistic regression is a **supervised machine learning algorithm** used for binary classification problems, where the output variable can take one of two values (e.g., benign or malignant cancer).
- It models the probability of the binary outcome using a logistic function, which is also known as the sigmoid function.
- Logistic regression is well-suited for problems where the goal is to predict the likelihood of a categorical outcome based on one or more predictor variables

The Dataset for Cancer Detection

- The dataset contains features computed from digitized images of fine needle aspirates (FNA) of breast masses, describing characteristics of the cell nuclei.
- The dataset was originally used for a Women Coders' Bootcamp organized by Artificial Intelligence for Development and UNDP Nepal, and is publicly available on the University of Wisconsin ftp server.
- The separating plane used to classify the data was obtained using the Multisurface Method-Tree (MSM-T), a linear programming-based classification technique.
- The dataset contains 569 instances with 30 features, and the target variable is the diagnosis (malignant or benign).
- The dataset has significant potential for social good, as it can contribute to the early and accurate diagnosis of breast cancer, and promote women's participation in AI and data science.
- https://www.kaggle.com/datasets/nancyalaswad90/breast-cancer-dataset/data
- https://github.com/rameshgitter/4th-Sem-Minor-Project

```python
import pandas as pd

# Load the dataset
data = pd.read_csv('cancer_dataset.csv')

# Display the first few rows of the dataset
print("First few rows of the dataset:")
print(data.head())

# Display the shape of the dataset
print("\nShape of the dataset:")
print(data.shape)

# Display the column names
print("\nColumn names:")
print(data.columns)

# Display the data types of the columns
print("\nData types:")
print(data.dtypes)

# Display summary statistics of the dataset
print("\nSummary statistics:")
print(data.describe())

# Display the distribution of the target variable
print("\nDistribution of the target variable:")
print(data['diagnosis'].value_counts())
```

Here's what this code does:

1. **Load the dataset**: The `pd.read_csv()` function is used to read the cancer dataset from a CSV file.
2. **Display the first few rows**: The `head()` method is used to display the first few rows of the dataset, giving you a quick glimpse of the data.
3. **Display the shape of the dataset**: The `shape` attribute is used to display the number of rows and columns in the dataset.
4. **Display the column names**: The `columns` attribute is used to display the names of the columns in the dataset.
5. **Display the data types**: The `dtypes` attribute is used to display the data types of the columns in the dataset.
6. **Display summary statistics**: The `describe()` method is used to display summary statistics, such as the count, mean, standard deviation, minimum, and maximum values for each numerical column.
7. **Display the distribution of the target variable**: The `value_counts()` method is used to display the distribution of the target variable (in this case, the 'diagnosis' column).

Output of above code

```
First few rows of the dataset:
         id diagnosis  ...  symmetry_worst  fractal_dimension_worst
0    842302         M  ...          0.4601                  0.11890
1    842517         M  ...          0.2750                  0.08902
2  84300903         M  ...          0.3613                  0.08758
3  84348301         M  ...          0.6638                  0.17300
4  84358402         M  ...          0.2364                  0.07678

[5 rows x 32 columns]

Shape of the dataset:
(569, 32)

Column names:
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave_points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave_points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave_points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

```
Data types:
id                         int64
diagnosis                 object
radius_mean              float64
texture_mean             float64
perimeter_mean           float64
area_mean                float64
smoothness_mean          float64
compactness_mean         float64
concavity_mean           float64
concave_points_mean      float64
symmetry_mean            float64
fractal_dimension_mean   float64
radius_se                float64
texture_se               float64
perimeter_se             float64
area_se                  float64
smoothness_se            float64
compactness_se           float64
concavity_se             float64
concave_points_se        float64
symmetry_se              float64
fractal_dimension_se     float64
radius_worst             float64
texture_worst            float64
perimeter_worst          float64
area_worst               float64
smoothness_worst         float64
compactness_worst        float64
concavity_worst          float64
concave_points_worst     float64
symmetry_worst           float64
fractal_dimension_worst  float64
dtype: object
```

What data types are present in dataset?

1. `int64` (64-bit Integer):

- The `int64` data type is used to represent integer values in pandas.
- It can store whole numbers within the range of -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
- `int64` is the default integer data type in pandas, unless the data requires a smaller range, in which case `int8`, `int16`, or `int32` may be used.
- Integer data types are efficient in terms of memory usage and computational performance, as they can be stored and processed more efficiently than other numerical data types.

2. `float64` (64-bit Floating-Point):

- The `float64` data type is used to represent floating-point (decimal) values in pandas.
- It can store numbers with decimal places, with a range of approximately 2.2250738585072014e-308 to 1.7976931348623157e+308.
- `float64` is the default numerical data type in pandas for any column that contains non-integer values.
- Floating-point numbers are useful for representing values that cannot be accurately expressed as integers, such as measurements, calculations, or statistical data.

3. `object` (Python Object):

- The `object` data type in pandas is used to represent arbitrary Python objects, including strings, lists, dictionaries, and other data structures.
- It is a flexible data type that can accommodate a wide range of data, but it is also less memory-efficient than the dedicated numeric data types (`int64` and `float64`).
- When working with text-based data, such as categorical variables or free-form text, the `object` data type is commonly used.
- However, it's often recommended to convert text-based data to more efficient data types, such as `category`, whenever possible, to optimize memory usage and performance.

```
Summary statistics:
                 id  radius_mean  ...  symmetry_worst  fractal_dimension_worst
count  5.690000e+02   569.000000  ...      569.000000               569.000000
mean   3.037183e+07    14.127292  ...        0.290076                 0.083946
std    1.250206e+08     3.524049  ...        0.061867                 0.018061
min    8.670000e+03     6.981000  ...        0.156500                 0.055040
25%    8.692180e+05    11.700000  ...        0.250400                 0.071460
50%    9.060240e+05    13.370000  ...        0.282200                 0.080040
75%    8.813129e+06    15.780000  ...        0.317900                 0.092080
max    9.113205e+08    28.110000  ...        0.663800                 0.207500

[8 rows x 31 columns]

Distribution of the target variable:
diagnosis
B    357
M    212
```

Sigmoid Function

- The sigmoid function is a crucial component of logistic regression, as it transforms the linear combination of the features and parameters into a probability value between 0 and 1.
- The sigmoid function is defined as: `1 / (1 + e^(-z))`, where `z` is the linear combination of the features and parameters.
- This transformation allows logistic regression to model the probability of the binary outcome, which can then be used to make classification decisions based on a chosen threshold (e.g., predict "malignant" if the probability is greater than 0.5).

Cost Function

- The cost function in logistic regression is the squared error cost, which measures the difference between the predicted probabilities and the actual target values.
- The cost function is defined as: `-(1/m) * (y.T * log(h) + (1 - y).T * log(1 - h))`, where `m` is the number of training examples, `y` is the target variable, and `h` is the predicted probability.
- The goal of the optimization process is to minimize this cost function, which will result in a model that best fits the training data.

Gradient Descent

- Gradient descent is the optimization algorithm used to find the optimal values of the model parameters (weights and bias) that minimize the cost function.
- The algorithm iteratively updates the parameters in the opposite direction of the gradients, scaled by the learning rate (α).
- The update rule for the parameters is: `theta = theta - (α/m) * X.T * (h - y)`, where `theta` are the model parameters, `X` are the feature inputs, and `h` and `y` are the predicted and actual target values, respectively.
- The learning rate (α) is a hyperparameter that needs to be carefully chosen to balance convergence speed and stability.

My Code:

- The libraries include `numpy` for numerical operations, `pandas` for data manipulation, `sklearn.model_selection` for train-test split, `sklearn.linear_model` for the logistic regression model, and `sklearn.metrics` for evaluation metrics.
- The dataset is loaded using the `pd.read_csv()` function from the pandas library.
- The features (`X`) and the target variable (`y`) are separated, where `y` represents the cancer diagnosis (e.g., benign or malignant).
- The dataset is then split into training and testing sets using the `train_test_split()` function from `sklearn.model_selection`.
- The `fit()` method is used to train the model by optimizing the model parameters to minimize the cost function.
- The `predict()` method is used to make predictions on the testing set, and various evaluation metrics are calculated, including accuracy, confusion matrix, and classification report.

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Load the dataset
data = pd.read_csv('cancer_dataset.csv')

# Separate the features and target variable
X = data.drop('diagnosis', axis=1)
y = data['diagnosis'].replace({'M': 1, 'B': 0})  # Convert target variable to numeric

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Sigmoid function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Squared error cost function
def cost_function(theta, X, y):
    m = len(y)
    h = sigmoid(np.dot(X, theta))
    return (-1/m) * (np.dot(y.T, np.log(h)) + np.dot((1 - y).T, np.log(1 - h)))

# Gradient descent
def gradient_descent(theta, X, y, alpha, num_iters):
    m = len(y)
    theta = theta.copy()

    for i in range(num_iters):
        h = sigmoid(np.dot(X, theta))
        theta = theta - (alpha/m) * np.dot(X.T, (h - y))

    return theta
```

Initializing Model Parameters & Setting Hyperparameters

The code starts by initializing the model parameters (`theta`) to a vector of zeros, with the length equal to the number of features (`X.shape[1]`).

The learning rate (`alpha`) is set to 0.01, which determines the step size for the gradient descent algorithm.

The number of iterations (`num_iters`) is set to 1000, which specifies the number of times the gradient descent algorithm will update the model parameters.

```
# Train the logistic regression model
initial_theta = np.zeros(X.shape[1])
alpha = 0.01
num_iters = 1000

theta = gradient_descent(initial_theta, X_train, y_train, alpha, num_iters)

# Make predictions on the test set
y_pred = sigmoid(np.dot(X_test, theta)) >= 0.5

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("Confusion Matrix:\n", cm)
print("Classification Report:\n", report)
```

Gradient Descent Optimization & Making Predictions on the Test Set

- The `gradient_descent()` function is called with the initial parameter values, training features (`X_train`), training target (`y_train`), learning rate, and number of iterations.
- This function implements the gradient descent algorithm to optimize the model parameters (`theta`) by minimizing the cost function.
- The gradient descent algorithm iteratively updates the parameters in the opposite direction of the gradients, scaled by the learning rate, until the specified number of iterations is reached.
- After training the model, the code uses the learned parameters (`theta`) to make predictions on the test set (`X_test`).
- The `sigmoid()` function is applied to the dot product of the test features and the learned parameters to obtain the predicted probabilities.
- These probabilities are then compared to a threshold of 0.5 to convert the probabilities into binary predictions (0 or 1).

Output Of My Code:

```
Accuracy: 0.6228070175438597
Confusion Matrix:
 [[71  0]
 [43  0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.62      1.00      0.77        71
           1       0.00      0.00      0.00        43

    accuracy                           0.62       114
   macro avg       0.31      0.50      0.38       114
weighted avg       0.39      0.62      0.48       114
```

Output Analysis & Classification Report:

- Precision: The precision for the benign class is 62%, meaning that 62% of the samples the model classified as benign were actually benign, which is a relatively low performance for a binary classification task.
- Recall: The recall for the benign class is 100%, meaning that the model correctly identified all the benign samples.
- F1-Score: The F1-score for the benign class is 77%, which is a balanced metric combining precision and recall.
- Confusion Matrix:
    a. True Positives (TP): 0 - The model correctly identified 0 samples as malignant.
    b. True Negatives (TN): 71 - The model correctly identified 71 samples as benign.
    c. False Positives (FP): 0 - The model incorrectly identified 0 samples as malignant.
    d. False Negatives (FN): 43 - The model incorrectly identified 43 samples as benign.
- The model performed poorly on the malignant class, with 0 precision, 0 recall, and 0 F1-score, indicating that it failed to correctly identify any of the malignant samples.

How to improve this?

- To improve the model's performance, the following steps can be considered:
  - Explore feature engineering techniques to provide more informative inputs to the model.
  - Investigate the use of regularization methods (e.g., L1 or L2 regularization) to address potential overfitting issues.
  - Experiment with different hyperparameter settings, such as the learning rate and the number of iterations for gradient descent.
  - Consider incorporating additional techniques, such as cross-validation, to better estimate the model's generalization capabilities.
  - Evaluate the need for a more complex model, such as a neural network, if the logistic regression model's performance remains unsatisfactory.

Modified code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Load the dataset
data = pd.read_csv('cancer_dataset.csv')

# Separate the features and target variable
X = data.drop('diagnosis', axis=1)
y = data['diagnosis']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("Confusion Matrix:\n", cm)
print("Classification Report:\n", report)
```

How this modified code working?

- The code starts by loading the cancer dataset from a CSV file using the `pd.read_csv()` function from the pandas library.
- The features (`X`) are obtained by dropping the 'diagnosis' column from the dataset.
- The target variable (`y`) is extracted from the 'diagnosis' column.
- The dataset is then split into training and testing sets using the `train_test_split()` function from `sklearn.model_selection`. The test size is set to 20% of the total data, and the `random_state` is set to 42 to ensure reproducibility.
- An instance of the `LogisticRegression()` model is created from the `sklearn.linear_model` module.
- The `fit()` method is used to train the logistic regression model on the training data (`X_train` and `y_train`).
- The trained model is used to make predictions on the testing set (`X_test`) using the `predict()` method, and the predicted values are stored in the `y_pred` variable.
- The performance of the model is evaluated using the following metrics:
    - Accuracy: Calculated using the `accuracy_score()` function from `sklearn.metrics`.
    - Confusion Matrix: Obtained using the `confusion_matrix()` function from `sklearn.metrics`.
    - Classification Report: Generated using the `classification_report()` function from `sklearn.metrics`.

Output of modified code

```
Accuracy: 0.956140350877193
Confusion Matrix:
 [[70  1]
 [ 4 39]]
Classification Report:
              precision    recall  f1-score   support

           B       0.95      0.99      0.97        71
           M       0.97      0.91      0.94        43

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
```

Entering a new Patient data and detecting cancer

```
# Generate a random input array with associated feature names
input_data = pd.DataFrame(np.random.rand(1, X.shape[1]), columns=X.columns)

# Make a prediction using the trained model
prediction = model.predict(input_data)

# Print the input data
print("New patient data:")
print(input_data)

# Interpret the prediction
if prediction[0] == 0:
    print("The patient does not have cancer.")
else:
    print("The patient has cancer.")
```

```
New patient data:
         id  radius_mean  texture_mean  perimeter_mean  area_mean  ...  compactness_worst  concavity_worst  concave_points_worst  symmetry_worst  fractal_dimension_worst
0  0.458182     0.848382      0.561362        0.884598   0.190959  ...           0.379086         0.626601              0.324492        0.939934                  0.13183

[1 rows x 31 columns]
The patient has cancer.
```

Thank you...