Name : Sagar Sidhu

Email : sagarsidhu33321@gmail.com

Assignment Name : Logistic Regression

Assignment code: DA-AG-011

Drive Link : N/A

Github Link : N/A

Question 1: What is Logistic Regression, and how does it differ from Linear Regression?

Answer=**Logistic Regression** is a statistical method used for **binary classification** problems, where the outcome variable is categorical (e.g., 0 or 1, true or false, spam or not spam). It estimates the **probability** that a given input belongs to a particular class using the **logistic (sigmoid) function** to map predicted values to a range between 0 and 1.

Key Differences from Linear Regression:

Aspect	**Linear Regression**	**Logistic Regression**
Output	Continuous value (e.g., 2.5, 100.7)	Probability (between 0 and 1)
Target variable	Quantitative (e.g., price, age)	Categorical (e.g., yes/no, 0/1)
Function used	Linear equation: $y = b_0 + b_1x$	Sigmoid function: p = 1 / (1 + e^(−(b_0 + b_1x)))
Goal	Predict the exact value of the target	Estimate the probability of class membership
Evaluation metrics	MSE, RMSE, R^2	Accuracy, Precision, Recall, ROC-AUC, Log-loss

Question 2: Explain the role of the Sigmoid function in Logistic Regression.

Answer=The **Sigmoid function** plays a central role in **Logistic Regression** by converting the output of a linear equation into a **probability value between 0 and 1**.

Sigmoid Function Formula:

σ(z)=1/1+e−z

Where:

- z=b0+b1x1+b2x2+⋯+bnxn (the linear combination of inputs)
- σ(z) is the predicted probability that the output belongs to class 1.

Role in Logistic Regression:

1. **Transforms Linear Output to Probability**:
   The raw output of the model (z) can range from −∞ to +∞, but probabilities must lie between 0 and 1. The sigmoid maps any real number to that range.
2. **Enables Binary Classification**:
   By applying a **threshold** (commonly 0.5), the model can decide the class:

- If σ(z)≥0.5, predict class **1**
- If σ(z)<0.5, predict class **0**

3. **Differentiable and Smooth**:
   The sigmoid function is smooth and differentiable, which is essential for optimization during model training using methods like **gradient descent**.

Visual Intuition:

- S-shaped curve
- As input becomes very negative, output → 0
- As input becomes very positive, output → 1

Question 3: What is Regularization in Logistic Regression and why is it needed?

Answer: **Regularization** in Logistic Regression is a technique used to **prevent overfitting** by **penalizing large coefficients** in the model. It adds a regularization term to the **loss function**, discouraging the model from becoming too complex or too sensitive to the training data.

Why Regularization is Needed:

1. **Prevents Overfitting**:
   Without regularization, the model may fit the training data too well, capturing noise rather than patterns. This reduces performance on unseen data.
2. **Improves Generalization**:
   By keeping weights small and simple, the model performs better on new, unseen examples.
3. **Controls Model Complexity**:
   Regularization helps balance bias and variance by penalizing overly complex models.

Types of Regularization in Logistic Regression:

Type	Regularization Term	Effect
L1 (Lasso)	(\lambda \sum	w_i

Type	Regularization Term	Effect
L2 (Ridge)	λ∑wi2	Shrinks weights smoothly, keeping all features but reducing their impact
Elastic Net	Combination of L1 and L2	Balances between sparsity and smoothness

---

Modified Loss Function (e.g., with L2):

Loss=−∑ylog(p)+(1−y)log(1−p)+λ∑wi2

Question 4: What are some common evaluation metrics for classification models, and why are they important?

Answer: Evaluation metrics for classification models are crucial because they help us **measure how well a model is performing**, especially when the data is **imbalanced** or the cost of different errors varies.

Common Evaluation Metrics:

1. **Accuracy**

Accuracy=TP +TN/TP + TN + FP + FN

   - Measures the overall correctness.
   - **Limitation**: Misleading if classes are imbalanced.

2. **Precision**

Precision=TP/TP + FP

   - Out of predicted positives, how many are actually positive.
   - Important when **false positives** are costly (e.g., spam detection).

3. **Recall (Sensitivity or True Positive Rate)**

Recall=TP/TP + FN

   - Out of actual positives, how many did the model identify.
   - Important when **false negatives** are costly (e.g., disease detection).

4. **F1-Score**

$F1 = 2 \times \text{Precision} \times \text{Recall} / \text{Precision} + \text{Recall}$

- Harmonic mean of precision and recall.
- Useful when you need a balance between precision and recall.

5. **Confusion Matrix**
    - A table showing **TP, TN, FP, FN**.
    - Helps visualize model performance and types of errors.
6. **ROC Curve & AUC (Area Under Curve)**
    - ROC plots True Positive Rate vs. False Positive Rate.
    - **AUC** represents the model's ability to distinguish between classes.
    - AUC = 1 is perfect, 0.5 means random guessing.

Why These Metrics Are Important:

- **Accuracy alone is not enough**, especially with imbalanced datasets.
- Different problems require different metrics — e.g., prioritize **recall** in medical diagnosis, **precision** in fraud detection.
- They guide model selection, tuning, and decision-making in real-world applications.

Question 5: Write a Python program that loads a CSV file into a Pandas DataFrame, splits into train/test sets, trains a Logistic Regression model, and prints its accuracy. (Use Dataset from sklearn package) (Include your Python code and output in the code box below.)
Answer:

```python
# Import necessary libraries
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load the dataset (e.g., breast cancer dataset)
data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target  # Add target column to DataFrame

# Split into features and target
X = df.drop('target', axis=1)
y = df['target']

# Split the 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)

# Create and train the Logistic Regression model
model = LogisticRegression(max_iter=10000)  # Increased max_iter for convergence
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of Logistic Regression model:", round(accuracy, 4))
```

```
Accuracy of Logistic Regression model: 0.9561
```

Question 6: Write a Python program to train a Logistic Regression model using L2 regularization (Ridge) and print the model coefficients and accuracy. (Use Dataset from sklearn package) (Include your Python code and output in the code box below.)

Answer:

```python
# Import required libraries
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load breast cancer dataset
data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

# Split features and target
X = df.drop('target', axis=1)
y = df['target']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train logistic regression model with L2 regularization (default penalty='l2')
model = LogisticRegression(penalty='l2', solver='liblinear', max_iter=1000)
model.fit(X_train, y_train)

# Predict on test set
y_pred = model.predict(X_test)

# Output model coefficients and accuracy
print("Model Coefficients:")
for feature, coef in zip(X.columns, model.coef_[0]):
    print(f"{feature}: {coef:.4f}")

# Print accuracy
accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy of Logistic Regression model with L2 regularization:", round(accuracy, 4))
```

```
Model Coefficients:
mean radius: 2.1325
mean texture: 0.1528
mean perimeter: -0.1451
mean area: -0.0008
mean smoothness: -0.1426
mean compactness: -0.4156
mean concavity: -0.6519
mean concave points: -0.3445
mean symmetry: -0.2076
mean fractal dimension: -0.0298
radius error: -0.0500
texture error: 1.4430
perimeter error: -0.3039
area error: -0.0726
smoothness error: -0.0162
compactness error: -0.0019
concavity error: -0.0449
concave points error: -0.0377
symmetry error: -0.0418
fractal dimension error: 0.0056
worst radius: 1.2321
worst texture: -0.4046
worst perimeter: -0.0362
worst area: -0.0271
worst smoothness: -0.2626
worst compactness: -1.2090
worst concavity: -1.6180
worst concave points: -0.6153
worst symmetry: -0.7428
worst fractal dimension: -0.1170

Accuracy of Logistic Regression model with L2 regularization: 0.9561
```

Question 7: Write a Python program to train a Logistic Regression model for multiclass classification using multi_class='ovr' and print the classification report. (Use Dataset from sklearn package) (Include your Python code and output in the code box below.)

Answer:

```python
# Import necessary libraries
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

# Load the iris dataset
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

# Features and target
X = df.drop('target', axis=1)
y = df['target']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train Logistic Regression model for multiclass classification using OvR (One-vs-Rest)
model = LogisticRegression(multi_class='ovr', solver='liblinear', max_iter=1000)
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Print classification report
print("Classification Report:\n")
print(classification_report(y_test, y_pred, target_names=data.target_names))
```

```
Classification Report:

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

Question 8: Write a Python program to apply GridSearchCV to tune C and penalty hyperparameters for Logistic Regression and print the best parameters and validation accuracy. (Use Dataset from sklearn package) (Include your Python code and output in the code box below.)

Answer:

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from warnings import simplefilter
simplefilter(action='ignore', category=FutureWarning)

# Load the Iris dataset
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

# Features and target
X = df.drop('target', axis=1)
y = df['target']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define parameter grid
param_grid = {
    'C': [0.01, 0.1, 1, 10],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear']  # liblinear supports both l1 and l2
}

# Set up GridSearchCV
grid = GridSearchCV(LogisticRegression(multi_class='ovr', max_iter=1000), param_grid, cv=5)
grid.fit(X_train, y_train)

# Best parameters and validation score
print("Best Parameters:", grid.best_params_)
print("Best Cross-Validation Accuracy:", round(grid.best_score_, 4))
```

```
# Test set performance
best_model = grid.best_estimator_
y_pred = best_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)
print("Test Set Accuracy:", round(test_accuracy, 4))
```

```
Best Parameters: {'C': 10, 'penalty': 'l1', 'solver': 'liblinear'}
Best Cross-Validation Accuracy: 0.9583
Test Set Accuracy: 1.0
```

Question 9: Write a Python program to standardize the features before training Logistic Regression and compare the model's accuracy with and without scaling. (Use Dataset from sklearn package) (Include your Python code and output in the code box below.)

Answer:

```
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load dataset
data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

# Features and target
X = df.drop('target', axis=1)
y = df['target']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# --- Without Scaling ---
model_raw = LogisticRegression(max_iter=10000)
model_raw.fit(X_train, y_train)
y_pred_raw = model_raw.predict(X_test)
accuracy_raw = accuracy_score(y_test, y_pred_raw)

# --- With Scaling ---
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model_scaled = LogisticRegression(max_iter=10000)
model_scaled.fit(X_train_scaled, y_train)
y_pred_scaled = model_scaled.predict(X_test_scaled)
accuracy_scaled = accuracy_score(y_test, y_pred_scaled)
```

```
# Print results
print("Accuracy WITHOUT Scaling:", round(accuracy_raw, 4))
print("Accuracy WITH Scaling   :", round(accuracy_scaled, 4))
```

```
Accuracy WITHOUT Scaling: 0.9561
Accuracy WITH Scaling   : 0.9737
```

Question 10: Imagine you are working at an e-commerce company that wants to predict which customers will respond to a marketing campaign. Given an imbalanced dataset (only 5% of customers respond), describe the approach you'd take to build a Logistic Regression model — including data handling, feature scaling, balancing classes, hyperparameter tuning, and evaluating the model for this real-world business use case.

Answer=

In this e-commerce use case, you're working with a **highly imbalanced dataset** where only **5% of customers respond** to the marketing campaign. Here's a structured approach to build a robust Logistic Regression model:

---

1. Data Understanding and Exploration

- **Inspect class imbalance**: Check percentage of positive (responders) vs. negative (non-responders).
- **Analyze features**: Identify categorical and numerical variables.
- **Handle missing values**, outliers, and data types appropriately.

---

2. Feature Engineering

- **Encode categorical variables**: Use one-hot encoding or ordinal encoding depending on the feature type.
- **Create new features**: Example — total spend in last 3 months, recency of last purchase, etc.
- **Remove low-variance or redundant features**.

---

3. Feature Scaling

- Apply **StandardScaler or MinMaxScaler** to numerical features because:
    - Logistic Regression is **sensitive to feature scales**.
    - Helps with model convergence and interpretability.

---

4. Handling Class Imbalance

Because only 5% of the dataset is positive (responders), you **must address imbalance** to avoid biased predictions.

Options:

- **Resampling Techniques**:
    - **Oversampling** the minority class (e.g., SMOTE)
    - **Undersampling** the majority class
    - Or a **combination of both**
- **Class weights**:
    - Use class_weight='balanced' in LogisticRegression to automatically adjust weights inversely proportional to class frequencies.

---

5. Model Training with Logistic Regression

python

CopyEdit

from sklearn.linear_model import LogisticRegression

model = LogisticRegression(class_weight='balanced', max_iter=1000)

model.fit(X_train_scaled, y_train)

---

6. Hyperparameter Tuning

Use **GridSearchCV** or **RandomizedSearchCV** to tune parameters like:

- C (regularization strength)
- penalty (l1, l2)
- solver (liblinear, saga depending on penalty)

---

7. Model Evaluation

Accuracy is **not reliable** in imbalanced problems. Instead, focus on:

Metric	**Why It's Important**
Precision	To avoid targeting uninterested users (false positives)
Recall	To identify as many interested users as possible (true positives)

Metric	Why It's Important
F1-score	Balances precision and recall
AUC-ROC	Measures the model’s ability to distinguish between classes
Confusion Matrix	Understand the distribution of predictions

---

8. Threshold Adjustment

- Logistic Regression outputs **probabilities**.
- The default threshold of 0.5 might not be optimal.
- Use techniques like **precision-recall trade-off** or **Youden’s J statistic** to select a custom threshold (e.g., 0.3).

---

9. Business Consideration

- **Cost-sensitive analysis**: Predicting a responder incorrectly (false negative) may lose a sale, while a false positive wastes marketing budget.
- Design a strategy that aligns model output with **ROI-based marketing decisions**.

Data Cleaning → Feature Engineering → Scaling → Resampling or Class Weighting →

Logistic Regression → Hyperparameter Tuning → Precision-Recall Evaluation → Threshold Optimization