Name: Abhishek Kushwaha

Email-id: askrocks2001@gmail.com

Assignment Name: Logistic Regression

Assignment Code: DA-AG-011

Github Link:

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

Answer:
Logistic Regression is a statistical method used to predict the probability of an event that has only two possible outcomes, such as yes/no, pass/fail, or 0/1. Instead of giving a direct numerical value like Linear Regression, it gives a probability score between 0 and 1, which can then be used to classify the data into categories.

The main difference between Logistic Regression and Linear Regression is in their purpose and output:

- **Linear Regression predicts continuous values (like predicting height, salary, or temperature).**
- **Logistic Regression predicts categorical outcomes (mainly binary), using a special function called the *sigmoid* to keep predictions within the range of 0 to 1.**

In short, Linear Regression is for continuous data, while Logistic Regression is for classification problems.

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

In Logistic Regression, the sigmoid function is used to turn any real-valued number into a value between 0 and 1. This is important because the output of Logistic Regression represents a probability, and probabilities must always be in that range.

The function has an "S" shaped curve, which makes it easy to map large negative numbers close to 0, large positive numbers close to 1, and values around zero to about 0.5.
By doing this, the sigmoid helps decide how likely it is that a given input

belongs to a particular class. If the probability is greater than a certain threshold (like 0.5), the model classifies it as one class; otherwise, it classifies it as the other.

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

Answer:
Regularization in Logistic Regression is a technique used to prevent the model from overfitting the training data. Overfitting happens when the model learns the noise and random patterns in the data instead of just the main trends, which makes it perform poorly on new, unseen data.

Regularization works by adding a penalty term to the model's cost function. This penalty discourages the model from assigning very large weights to features, which helps keep the model simpler and more general.

It is needed because without regularization, especially when there are many features, the model can become too complex and fit the training data too closely, losing its ability to make accurate predictions on fresh data.

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

Answer:
Some common evaluation metrics for classification models are:

1. **Accuracy – The percentage of correct predictions out of all predictions. It's simple to understand but can be misleading if the data is imbalanced.**
2. **Precision – Out of all the positive predictions the model made, how many were actually correct. Useful when the cost of false positives is high.**

3. **Recall (Sensitivity) – Out of all actual positives, how many did the model correctly identify. Important when missing a positive case is costly.**
4. **F1-Score – The harmonic mean of precision and recall. It balances both metrics, especially in cases of uneven class distribution.**
5. **ROC-AUC Score – Measures how well the model can distinguish between classes, regardless of the classification threshold.**

These metrics are important because they help you understand not just *how often* the model is correct, but also *how* it is making mistakes, which is critical for improving performance.

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.

```
# Importing required 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 accuracy_score


# Load dataset from sklearn (Iris dataset for example)

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
```

```
df['target'] = iris.target


# Selecting only two classes for binary classification
df = df[df['target'] != 2]


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


# Train-test split (80% training, 20% testing)
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
model = LogisticRegression()
model.fit(X_train, y_train)


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

```
# Calculate and print accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Model Accuracy:", accuracy)
```

Question 6: Write a Python program to train a Logistic Regression model using L2 regularization (Ridge) and print the model coefficients and accuracy.

Answer:

```
# Import 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 accuracy_score

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

```
# Keep only two classes for binary classification
df = df[df['target'] != 2]

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

# Split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Logistic Regression with L2 regularization
model = LogisticRegression(penalty='l2', solver='liblinear')
model.fit(X_train, y_train)

# Model coefficients
print("Model Coefficients:", model.coef_)
print("Intercept:", model.intercept_)

# Predictions and accuracy
y_pred = model.predict(X_test)
```

accuracy = accuracy_score(y_test, y_pred)

print("Model Accuracy:", accuracy)

sample output:

Model Coefficients: [[0.39 -0.91 2.32 -2.17]]

Intercept: [-0.12]

Model Accuracy: 1.0

Question 7: Write a Python program to train a Logistic Regression model for multiclass classification using multi_class='ovr' and print the classification report.

Answer:

 # Import 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 Iris dataset

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

```
df['target'] = iris.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)

# Logistic Regression for multiclass classification
model = LogisticRegression(multi_class='ovr', solver='liblinear')
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Print classification report
print(classification_report(y_test, y_pred))
```

sample output:

```
              precision    recall  f1-score   support
```

0	1.00	1.00	1.00	10
1	1.00	1.00	1.00	9
2	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.

Answer:

```
# Import libraries
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

# Load dataset
iris = load_iris()
```

```
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

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

# Split into train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Logistic Regression model
model = LogisticRegression(solver='liblinear')

# Parameter grid for tuning
param_grid = {
   'C': [0.1, 1, 10],
   'penalty': ['l1', 'l2']
}

# Apply GridSearchCV
grid = GridSearchCV(model, param_grid, cv=5)
```

```
grid.fit(X_train, y_train)

# Print best parameters and validation accuracy
print("Best Parameters:", grid.best_params_)
print("Validation Accuracy:", grid.best_score_)
```

sample output:

```
Best Parameters: {'C': 1, 'penalty': 'l2'}
Validation Accuracy: 0.9666666666666668
```

Question 9: Write a Python program to standardize the features before training Logistic Regression and compare the model's accuracy with and without scaling.

Answer:

```
# Import 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.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
```

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

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

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Logistic Regression without scaling
model_no_scale = LogisticRegression(max_iter=200)
model_no_scale.fit(X_train, y_train)
pred_no_scale = model_no_scale.predict(X_test)
acc_no_scale = accuracy_score(y_test, pred_no_scale)

# Standardize features
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Logistic Regression with scaling
model_scaled = LogisticRegression(max_iter=200)
model_scaled.fit(X_train_scaled, y_train)
pred_scaled = model_scaled.predict(X_test_scaled)
acc_scaled = accuracy_score(y_test, pred_scaled)

# Print comparison
print("Accuracy without Scaling:", acc_no_scale)
print("Accuracy with Scaling   :", acc_scaled)
```

sample output:

Accuracy without Scaling: 0.9666666666666667

Accuracy with Scaling   : 1.0

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:
For this problem, I would follow these steps to make sure the Logistic Regression model works well despite the data being imbalanced:

1. **Understand and clean the data – First, I'd explore the dataset to check for missing values, outliers, or irrelevant features, and fix or remove them so the data is reliable.**
2. **Feature scaling – Since Logistic Regression is affected by the scale of features, I'd standardize them using something like `StandardScaler` so that all features contribute fairly to the model.**
3. **Handling class imbalance – With only 5% positive responses, the model could easily get biased towards predicting "no response." To handle this:**
    - **I'd try oversampling the minority class using SMOTE or undersampling the majority class.**
    - **Another option is to use the `class_weight='balanced'` parameter in Logistic Regression to give more importance to the minority class.**
4. **Model training – I'd train the Logistic Regression model with regularization (L1 or L2) to avoid overfitting, and set `max_iter` high enough to ensure convergence.**
5. **Hyperparameter tuning – I'd use `GridSearchCV` or `RandomizedSearchCV` to find the best values for `C` (regularization strength), `penalty` type, and class weights.**
6. **Evaluation metrics – Since accuracy can be misleading for imbalanced data, I'd focus on metrics like Precision, Recall, F1-score, ROC-AUC, and the confusion matrix. In this business case, high recall might be important to ensure we reach most potential responders, even at the cost of some false positives.**
7. **Business perspective – I'd also work with the marketing team to set the right decision threshold based on cost-benefit analysis. For example, if the cost of contacting a non-responder is low compared to missing a real responder, we can lower the threshold to catch more positives.**

This approach ensures that the model is fair, well-tuned, and aligned with the company's goal of maximizing campaign success