Assignment Code: DA-AG-011

Logistic Regression | **Assignment**

Instructions: Carefully read each question. Use Google Docs, Microsoft Word, or a similar tool to create a document where you type out each question along with its answer. Save the document as a PDF, and then upload it to the LMS. Please do not zip or archive the files before uploading them. Each question carries 20 marks.

Total Marks: 200

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

Answer:

Logistic Regression is a statistical model used for binary or multiclass classification, which estimates the probability that a given input belongs to a particular category. It uses the **Sigmoid function** to map predicted values to probabilities between 0 and 1.

How does it differ from Linear Regression ?

Feature	**Linear Regression**	**Logistic Regression**
Definition	Predicts a **continuous numeric value** based on input features	Predicts the **probability of a class** based on input features
Purpose	Regression problems	Classification problems
Output Range	Any real number	Between 0 and 1
Function Used	Linear equation	Sigmoid (logistic) function
Use Case	Predicting prices, scores, etc.	Predicting categories (e.g., spam vs. not spam, pass vs fail, diabetic vs non-diabetic)

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

Answer: Role of Sigmoid Function in Logistic Regresion:

The **Sigmoid function** maps any real-valued number, to a value between (0,1), making it ideal for expressing **probabilites**.

Formula: $\sigma(z) = \frac{1}{1 + e^{-z}}$

- $\sigma(z)$ is the sigmoid function applied to input z
- e^{-z} is the exponential decay term
- The output is always between 0 and 1

Role in Logistic Regression:

- Converts the linear combination of inputs (z) into a probability.
- Helps classify inputs by applying a threshold (e.g., $\geq 0.5 \rightarrow$ class 1, $< 0.5 \rightarrow$ class 0).
- Ensures output is interpretable as a likelihood of belonging to a class.

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

Answer: Regularization is a technique used to prevent overfitting by adding a penalty term to the loss function, discouraging overly complex models.

Why It's Needed:

- Helps control model complexity by shrinking large weights.
- Improves generalization to unseen data.
- Reduces variance without significantly increasing bias.

Types Commonly Used:

- L1 Regularization (Lasso): Adds $\lambda \sum |w_i|$ to the loss function. Encourages sparsity.
- L2 Regularization (Ridge): Adds $\lambda \sum w_i^2$ to the loss. Penalizes large weights smoothly.

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

Answer: Some common Evaluation Metrics for Classification Models:

Metric	Description	Why It's Important
Accuracy	Proportion of correct predictions	Good for balanced datasets
Precision	(\frac{\text{TP}}{\text{TP} + \text{FP}}) — True Positives over predicted positives	Measures exactness (low false positives)
Recall	(\frac{\text{TP}}{\text{TP} + \text{FN}}) — True Positives over actual positives	Measures completeness (low false negatives)
F1 Score	Harmonic mean of precision and recall	Balances precision and recall
ROC-AUC	Area under the Receiver Operating Characteristic curve	Evaluates model's ability to distinguish classes

Why They Matter:

- Different metrics highlight different aspects of performance.
- Crucial for imbalanced datasets (e.g., fraud detection).
- Help choose the best model for the specific business or domain need.

Extended Evaluation Tools

- Accuracy Score: Direct metric computed via accuracy_score(y_true, y_pred).
- Confusion Matrix: Tabular summary of TP, TN, FP, FN — foundation for other metrics.
- Classification Report: Consolidated view of precision, recall, F1-score, and support per class.

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: We will be using breast_cancer_data from sklearn.datasets.

In [6]:
```
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, confusion_matrix, classification_r
import warnings
warnings.filterwarnings('ignore')

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

# Since the data is pretty clean with no null or missing values.
# and all columns are float. we will proceed with Train_test_split.

# Splitting into train test sets.
X = df.drop('target', axis=1)
y = df.target

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando

# Train Logistic Reg. model.
model = LogisticRegression(max_iter =1000 )   #  max_iter=1000 ensures converg
model.fit(x_train, y_train)

# Predict
y_pred = model.predict(x_test)

# Print accuracy
accuracy = accuracy_score(y_test, y_pred)
print ("Accuracy:", round(accuracy * 100, 2), "%")

# Evaluation metrics.

print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

#Insights.: The model has an accuracy of 95.61 %
# - The model performs very well, especially in identifying malignant tumors (
# - False negatives are low, which is crucial in healthcare.
# Precision: Of all predicted malignant cases, how many were truly malignant?
# - High precision (e.g., 0.95) → few false positives.
# Recall: Of all actual malignant cases, how many were correctly identified?
# - High recall (e.g., 0.99) → few false negatives.
# F1-score: Harmonic mean of precision and recall.
# - Balances both metrics, especially useful when classes are imbalanced.
```

```
Accuracy: 95.61 %

Confusion Matrix:
 [[39  4]
 [ 1 70]]

Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.91      0.94        43
           1       0.95      0.99      0.97        71

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

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: We will be using breast cancer data for Logistic Regression with L2 (Ridge) Regularization.

In [7]:

```python
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# Load dataset
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)

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

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

# Evaluate
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

# Print results
print("Model Accuracy:", round(accuracy * 100, 2), "%")
print("\nModel Coefficients:")
for feature, coef in zip(X.columns, model.coef_[0]):
```

```
    print(f"{feature}: {coef:.4f}")

#Note:
# - penalty='l2' is the default, but explicitly specifying it makes the intent
# - solver='liblinear' is compatible with small datasets and L2 regularization
# - Coefficients show the weight each feature contributes to the decision boun
```

```
Model Accuracy: 95.61 %

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
```

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: Multiclass Logistic Regression with multi_class='ovr' using iris dataset.

In [8]:
```
import pandas as pd
```

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

# Load multiclass dataset
data = load_iris()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)

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

# Train Logistic Regression with One-vs-Rest strategy
model = LogisticRegression(multi_class='ovr', solver='liblinear')
model.fit(X_train, y_train)

# Predict and evaluate
y_pred = model.predict(X_test)
print("Accuracy: ", round(accuracy * 100, 2), "%")
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred, target_names=data.target_names))

# Key Insights:
# - multi_class='ovr' trains one binary classifier per class.
# - solver='liblinear' supports OvR and small datasets.
# - The classification report includes precision, recall,
# and F1-score for each class (setosa, versicolor, virginica).
```

```
Accuracy:  95.61 %

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: Hyperparameter Tuning with GridSearchCV

In [9]:
```
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import accuracy_score

# Load dataset
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)

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

# Define parameter grid
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2']
}

# Initialize model
model = LogisticRegression(solver='liblinear')  # liblinear supports both l1 a

# Grid search
grid = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
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_ * 100, 2), "%"

# Evaluate on test set
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 * 100, 2), "%")

#  Insights:
# - C controls regularization strength: lower values = stronger regularization
# - penalty: 'l1' for sparse models, 'l2' for ridge-style regularization.
# - solver='liblinear' is required for 'l1' penalty.
```

```
Best Parameters: {'C': 100, 'penalty': 'l1'}
Best Cross-Validation Accuracy: 96.7 %
Test Set Accuracy: 98.25 %
```

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: Here is the code that:

- Loads a dataset from sklearn
- Trains Logistic Regression with and without feature scaling
- Compares the accuracy of both models

In [10]:

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

# Load dataset
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)

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

# --- Model without scaling ---
model_raw = LogisticRegression(max_iter=1000)
model_raw.fit(X_train, y_train)
y_pred_raw = model_raw.predict(X_test)
accuracy_raw = accuracy_score(y_test, y_pred_raw)

# --- Model with scaling ---
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model_scaled = LogisticRegression(max_iter=1000)
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)

# --- Results ---
print("Accuracy without Scaling:", round(accuracy_raw * 100, 2), "%")
print("Accuracy with Scaling   :", round(accuracy_scaled * 100, 2), "%")


# Insights:
# - Standardization rescales features to have mean 0 and variance 1.
```

```
# - Logistic Regression (especially with regularization) benefits from scaling
# - There's a noticeable improvement in accuracy with scaling, especially when
```

```
Accuracy without Scaling: 95.61 %
Accuracy with Scaling   : 97.37 %
```

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: This is an imbalanced classification problem using Logistic Regression in an e-commerce setting:

Business Scenario

- Goal: Predict which customers will respond to a marketing campaign
- Challenge: Only 5% of customers respond → highly imbalanced dataset

Step-by-Step Approach

1. Data Handling

- Explore & clean: Handle missing values, outliers, and categorical encoding (e.g., one-hot or label encoding).
- Feature engineering: Create meaningful features like:
- Recency, frequency, monetary value (RFM)
- Past campaign interactions
- Customer segmentation tags

2. Feature Scaling

- Apply StandardScaler to normalize numerical features.
- Logistic Regression is sensitive to feature magnitude, especially with regularization.

3. Class Balancing Since only 5% respond:

- Resampling techniques:
- SMOTE (Synthetic Minority Over-sampling Technique) to generate synthetic positives.

- Random undersampling of majority class (optional).
- Class weights:
- Use class_weight='balanced' in LogisticRegression to penalize misclassification of minority class.

4. Model Training:

```
 model = LogisticRegression(class_weight='balanced',
solver='liblinear')
```

- Use solver='liblinear' for small datasets and L1/L2 regularization support.

5. Hyperparameter Tuning

- Use GridSearchCV to tune:
  - C: Regularization strength
  - penalty: 'l1' or 'l2'
  - class_weight: 'balanced' vs custom weights

6. Evaluation Metrics:

- Accuracy is misleading here. Focus on:

Metric	Why It Matters
Precision	Avoid false positives (wasting marketing budget)
Recall	Capture as many responders as possible
F1 Score	Balance between precision and recall
ROC-AUC	Overall ability to distinguish responders

- Use confusion matrix to monitor false negatives (missed responders).
- Consider Precision-Recall curve for better insight in imbalanced settings.