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 supervised machine learning algorithm used for classification problems, especially when the target variable is categorical (often binary, like "yes/no", "spam/not spam").

| Feature | Linear Regression | Logistic Regression |
|---|---|---|
| **Goal** | Predicts a **continuous** value | Predicts a **probability** (classification) |

| | | |
|---|---|---|
| **Output Range** | (−∞,+∞)(-\infty, +\infty) | 00 to 11 |
| **Model Equation** | y=β0+β1x1+...y = \beta_0 + \beta_1x_1 + ... | P(y=1)=11+e−zP(y=1) = \frac{1}{1 + e^{-z}} where z=β0+β1x1+...z = \beta_0 + \beta_1x_1 + ... |
| **Error Measurement** | Mean Squared Error (MSE) | Log Loss (Cross-Entropy) |
| **Assumption** | Linear relationship between variables | Log-odds have a linear relationship with predictors |
| **Application** | Price prediction, demand forecasting | Fraud detection, classification problems |

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

**Answer:**

### Role of Sigmoid Function in Logistic Regression

- **Transforms**: Converts any real-valued number into a probability-like value.
- **Probability Interpretation**: Output is interpreted as P(y=1∣x)P(y=1|x)P(y=1∣x).
- **Smooth Decision Boundary**: Creates an **S-shaped curve** instead of a straight line, allowing better separation of classes.
- **Mathematical Link**: Makes the model work with the **log-odds** of the target variable.

**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 adding a penalty term to the model's cost function.

It ensures that the model not only fits the training data well but also generalizes better to unseen data.

---

#### Why We Need Regularization

- Logistic Regression learns coefficients (β\betaβ) for each feature.
- If some coefficients become **too large**, the model becomes overly sensitive to small changes in input → **overfitting**.
- Overfitting means:

- High accuracy on training data.
- Poor accuracy on test data.

- Regularization **shrinks large coefficients**, making the model simpler and more robust.

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

**Answer:**

## 1. Common Evaluation Metrics

### (a) Accuracy

Accuracy = Total Predictions/Correct Predictions

### (b) Precision

Precision = True Positive/True Positives + False Positives

### (c) Recall (Sensitivity / True Positive Rate)

Recall = True Positives/True Positives + False Negatives

### (d) F1-Score

F1 = 2×Precision × Recall/Precision + Recall

### (e) Confusion Matrix

**(f) ROC Curve & AUC (Area Under Curve)**

## 2. Why These Metrics Are Important

- **Accuracy alone can mislead when classes are imbalanced.**
- **Precision & Recall give insight into different types of errors.**
- **F1-Score balances precision and recall.**
- **Confusion Matrix shows exactly where the model is making mistakes.**
- **AUC-ROC gives a threshold-independent view of performance.**
- **Log Loss considers the confidence of predictions**

**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:**

**Code:**

```
 # Logistic Regression Example with sklearn Dataset

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

# 1. Load dataset from sklearn
data = load_breast_cancer()
```

```
# 2. Create a DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

print("First 5 rows of dataset:")
print(df.head())

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

# 4. Train/Test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 5. Create Logistic Regression model
model = LogisticRegression(max_iter=5000)  # Increased max_iter for convergence
model.fit(X_train, y_train)

# 6. Make predictions
y_pred = model.predict(X_test)

# 7. Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

print(f"\nModel Accuracy: {accuracy:.4f}")
```

**Output:**

```
First 5 rows of dataset:
   mean radius  mean texture  mean perimeter  ...  worst symmetry  worst fractal dimension
target
0       17.99        10.38         122.80  ...         0.4601                 0.11890       0
1       20.57        17.77         132.90  ...         0.2750                 0.08902       0
2       19.69        21.25         130.00  ...         0.3613                 0.08758       0
3       11.42        20.38          77.58  ...         0.6638                 0.17300       0
4       20.29        14.34         135.10  ...         0.2364                 0.07678       0

Model Accuracy: 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:**

```
# Logistic Regression with L2 Regularization (Ridge)

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

# 1. Load dataset
data = load_breast_cancer()

# 2. Create DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

print("First 5 rows of dataset:")
print(df.head())

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

# 4. Train/Test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 5. Create Logistic Regression model with L2 regularization
model = LogisticRegression(penalty='l2', solver='lbfgs', max_iter=5000)  # L2 is default
model.fit(X_train, y_train)

# 6. Model coefficients
```

```
coefficients = pd.Series(model.coef_[0], index=X.columns)

# 7. Make predictions & calculate accuracy
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

# 8. Output results
print("\nModel Coefficients (L2 Regularization):")
print(coefficients)

print(f"\nModel Accuracy: {accuracy:.4f}")
```

**Output:**

```
First 5 rows of dataset:
   mean radius  mean texture  mean perimeter  ...  worst symmetry  worst fractal
dimension  target
0       17.99         10.38          122.80  ...          0.4601                0.11890       0
1       20.57         17.77          132.90  ...          0.2750                0.08902       0
2       19.69         21.25          130.00  ...          0.3613                0.08758       0
3       11.42         20.38           77.58  ...          0.6638                0.17300       0
4       20.29         14.34          135.10  ...          0.2364                0.07678       0

Model Coefficients (L2 Regularization):
mean radius                0.005527
mean texture               0.017735
mean perimeter             0.000931
mean area                  0.000651
mean smoothness           -0.289051
...
worst symmetry            -0.056171
worst fractal dimension   -0.035784
dtype: float64

Model Accuracy: 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:**

```
# Logistic Regression for Multiclass Classification (One-vs-Rest)

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

# 1. Load Iris dataset
data = load_iris()

# 2. Create DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

print("First 5 rows of dataset:")
print(df.head())

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

# 4. Train/Test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 5. Create Logistic Regression model (One-vs-Rest)
model = LogisticRegression(multi_class='ovr', solver='lbfgs', max_iter=200)
model.fit(X_train, y_train)

# 6. Predictions
y_pred = model.predict(X_test)

# 7. Classification report
report = classification_report(y_test, y_pred, target_names=data.target_names)

# 8. Output results
print("\nClassification Report:")
print(report
```

```
Output:

First 5 rows of dataset:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  target
0                5.1               3.5                1.4               0.2       0
1                4.9               3.0                1.4               0.2       0
2                4.7               3.2                1.3               0.2       0
3                4.6               3.1                1.5               0.2       0
4                5.0               3.6                1.4               0.2       0

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      0.90      0.95        10
   virginica       0.91      1.00      0.95        10

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        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 for Logistic Regression using GridSearchCV

import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
```

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

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

# 3. Train/Test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 4. Define Logistic Regression model
model = LogisticRegression(solver='liblinear', max_iter=1000)

# 5. Define parameter grid
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],        # Regularization strength
    'penalty': ['l1', 'l2']              # L1 = Lasso, L2 = Ridge
}

# 6. Apply GridSearchCV
grid = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
grid.fit(X_train, y_train)

# 7. Print best parameters and score
print("Best Parameters:", grid.best_params_)
print(f"Best Cross-Validation Accuracy: {grid.best_score_:.4f}")
```

**Output:**

**Best Parameters: {'C': 1, 'penalty': 'l1'}**
**Best Cross-Validation Accuracy: 0.9560**

**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:**

```
# Logistic Regression Accuracy Comparison: With vs Without Feature Scaling

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.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

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

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

# 3. Train/Test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# ------------------------------
# Model without scaling
# ------------------------------
model_no_scaling = LogisticRegression(max_iter=5000)
model_no_scaling.fit(X_train, y_train)
y_pred_no_scaling = model_no_scaling.predict(X_test)
accuracy_no_scaling = accuracy_score(y_test, y_pred_no_scaling)

# ------------------------------
# Model with scaling
```

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

model_with_scaling = LogisticRegression(max_iter=5000)
model_with_scaling.fit(X_train_scaled, y_train)
y_pred_with_scaling = model_with_scaling.predict(X_test_scaled)
accuracy_with_scaling = accuracy_score(y_test, y_pred_with_scaling)

# -------------------------------
# Print results
# -------------------------------
print(f"Accuracy without Scaling: {accuracy_no_scaling:.4f}")
print(f"Accuracy with Scaling   : {accuracy_with_scaling:.4f}")
```

**Output:**

**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:**

## 1. Problem Understanding

- **Goal:** Predict if a customer will respond to a marketing campaign (binary classification).

- **Challenge:** Only **5% positive class (responders) → highly imbalanced dataset**.
- **Impact:** Standard accuracy will be misleading — a model predicting “No” for everyone would get 95% accuracy but be useless.

---

# 2. Data Handling

## a. Data Cleaning

- Remove duplicates, handle missing values.
- Correct data types (dates → datetime, categories → categorical).
- Detect and fix anomalies (e.g., negative purchase amounts).

## b. Feature Engineering

- Create meaningful features:
    - **RFM metrics**: Recency (days since last purchase), Frequency (purchase count), Monetary value.
    - Campaign interaction history.
    - Customer demographics.
    - Web/app engagement metrics.
- Encode categorical variables (One-Hot Encoding for Logistic Regression).

---

# 3. Feature Scaling

- Logistic Regression is sensitive to feature scale.
- Apply **StandardScaler** (z-score normalization) after splitting into train/test sets.

---

## 4. Handling Class Imbalance

Since positives are rare (5%), I'd try:

**Class Weight Adjustment** (first choice for Logistic Regression):

```
LogisticRegression(class_weight='balanced')
```

1. → Penalizes mistakes on the minority class more.
2. **Oversampling Minority Class** (e.g., SMOTE) or **undersampling majority class**.
3. Possibly combine both (hybrid sampling) if dataset size is small.

---

## 5. Model Building

Base model:

```
LogisticRegression(
    penalty='l2',          # Ridge regularization
    solver='liblinear',    # Works with small datasets + class weights
    class_weight='balanced',
    max_iter=5000
)
```

-

- Train on scaled features.

---

## 6. Hyperparameter Tuning

Use **GridSearchCV** or **RandomizedSearchCV** over:

- **C** (inverse of regularization strength).
- **Penalty** (`l1` or `l2`).
- Possibly different solvers (`liblinear`, `saga`).

Example parameter grid:

```
param_grid = {
    'C': [0.01, 0.1, 1, 10],
    'penalty': ['l1', 'l2']
}
```

- Scoring metric: **ROC-AUC** (better for imbalanced data than accuracy).

---

## 7. Model Evaluation

For imbalanced datasets, prioritize:

- **Precision**: How many predicted responders are actual responders? (Important to avoid wasting campaign costs.)
- **Recall**: How many actual responders did we catch? (Important for maximizing campaign reach.)

- **F1-score**: Balances precision & recall.
- **ROC-AUC**: Measures model's ranking ability.
- **PR-AUC** (Precision-Recall Curve): More informative than ROC-AUC for high imbalance.
- **Confusion Matrix**: To visualize TP, FP, TN, FN.

---

# 8. Threshold Tuning

- Logistic Regression outputs probabilities → default 0.5 threshold might not be optimal.
- Tune the probability threshold to maximize a **business-specific metric**:
  - If cost of false positives is high → increase threshold.
  - If cost of false negatives is high → decrease threshold.
- Example: Optimize for maximum profit or minimum cost.

---

# 9. Deployment Considerations

- Retrain model periodically — campaign response behavior can drift.
- Monitor:
  - Model performance (ROC-AUC over time).
  - Class distribution changes.

- Calibration of predicted probabilities.

---

✅ **Summary Approach:**

1. Clean & engineer customer features.
2. Scale numeric features.
3. Handle class imbalance (class weights / SMOTE).
4. Build Logistic Regression model with regularization.
5. Tune hyperparameters with ROC-AUC as scoring.
6. Evaluate using precision, recall, F1, ROC-AUC, PR-AUC.
7. Adjust probability threshold for best business trade-off.
8. Deploy & monitor model.