Logistic Regression | **Assignment**

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

Answer:

Logistic regression and linear regression are both statistical models, but they are used for different types of problems. Linear regression predicts a continuous outcome, while logistic regression predicts a categorical outcome, typically binary (0 or 1).

More detailed breakdown:

**Linear Regression**:

1). Purpose: Predicts a continuous dependent variable (e.g., house price, temperature) based on independent variables.

2). Output: A continuous numerical value.

3). Example: Predicting the price of a house based on its size, location, and number of bedrooms.

4). Method: Uses the least squares method to find the best-fitting line.

5). Key Assumption: Linear relationship between the independent and dependent variables.

**Logistic Regression**:

1). Purpose: Predicts the probability of a binary outcome (e.g., spam/not spam, pass/fail) based on independent variables.

2). Output: A probability value between 0 and 1, which is often converted to a binary classification (e.g., above 0.5 is 1, below 0.5 is 0).

3). Example: Predicting whether a customer will click on an advertisement based on their demographics and browsing history.

4). Method: Uses maximum likelihood estimation to find the best-fitting sigmoid curve.

5). Key Assumption: The relationship between the independent variables and the log-adds of the outcome is linear.

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

Answer: In logistics regression, the sigmoid function acts as a crucial bridge, transforming the output of a linear model into a probability value between 0 and 1, which is then used for binary classification. This function ensures that the model’s output, representing the likelihood of an event, stays within the acceptable probability range.

More Detailed Explanation:

Linear Regression Output: Logistic regression begins with a linear equation, similar to linear regression, that combines input features with learned weights.

Sigmoid Transformation: The sigmoid function then takes the linear output and squashes it into a value between 0 and 1. This is done using the formula: g(z) = 1 / (1 + exp(-z) ), where ‘z’ is the output of the linear equation.

Probability Interpretation: The output of the sigmoid function, ranging from 0 to 1, represents the probability that a given input belongs to the positive class.

Classification Decision: A threshold (usually 0.5) is then applied to this probability. If the probability is above the threshold, the input is classified as belonging to the positive class (e.g., “yes”, “spam”, “malignant”). If it’s below, it’s classified as belonging to the negative class.

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

Answer: Regularization in logistic regression is a technique used to prevent overfitting by adding a penalty term to the loss function. This penalty discourages the model from fitting the training data too closely, which can lead to poor generalization on unseen data. Without regularization, logistic regression models, particularly with high-dimensional data, can become overly complex and fit noise in the training set, resulting in poor performance on new data.

Why is it needed?

1). Overfitting: Logistic regression, especially with a large number of features, can easily overfit the training data. This means the model learns the training data’s noise and random fluctuations instead of the underlying patterns, leading to poor generalization.

2). High-dimensional data: In high-dimensional spaces, it’s easy for logistic regression to find complex decision boundaries that perfectly separate the training data, but these boundaries might not generalize well.

3). Loss function behaviour: Logistic regression, without regularization, tends to drive the loss function towards zero, potentially leading to extremely large coefficient values, further exacerbating overfitting.

Common types of regularization in logistics regression:

L1 regularization (Ridge):

Adds a penalty proportional to the absolute value of the coefficients. This can lead to some coefficients being shrunk to exactly zero, effectively performing feature selection.

L2 regularization (Ridge):

Adds a penalty proportional to the square of the coefficients. This shrinks the coefficients towards zero but typically doesn’t make them exactly zero.

Elastic Net regularization:

Combines L1 and L2 regularization, offering a balance between feature selection and coefficient shrinkage.

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

Answer: Common classification model evaluation metrics include accuracy, precision, recall, F1-score, and AUC-ROC, each offering a different perspective on model performance.

These metrics are vital for assessing how well a model generalizes to new data and for identifying areas where the model might be underperforming, particularly in imbalanced datasets.

Breakdown of these metrics:

1). Accuracy: The proportion of correctly classified instances out of the total instances.

Formula : (True Positives + True Negatives) / Total Instances

Examples: If a model correctly identifies 90 out of 100 instances (either positive or negative), its accuracy is 90%.

2). Precision: The proportion of correctly predicted positive instances out of all instances predicted as positive.

Formula: True Positives / (True Positives + False Positives).

Example: In a spam detection model, high precision means that when the model flags an email as spam, it's likely actually spam (low false positives).

3). [Recall](https://www.google.com/search?sca_esv=e29194d70e62ae1b&cs=1&sxsrf=AE3TifNyIrXo0Yimy163H60bS-aattw3rw%3A1753953347619&q=Recall&sa=X&ved=2ahUKEwiUmc6x4eaOAxUo3TgGHSB3GToQxccNegQIOhAC&mstk=AUtExfBT-Sivlx_STngzLFUKWRpeNvRYoLie-6HN6jb3Xbzu7nHQimdBV81gKTFQ--S46rbX2wIr-oMuNtb2Bm_1t7ki626LG1I3ezhOrFJKtd0ESeZoJi3XoxG4zoEtiniyIFwxypJwOrk03FzTa3UWYwwSng0QN41x8OMkO60kKycpGXI&csui=3) (also known as Sensitivity or True Positive Rate): The proportion of correctly predicted positive instances out of all actual positive instances.

Formula: True Positives / (True Positives + False Negatives).

Examples: In a medical diagnosis model, high recall means the model identifies most patients with a disease (low false negatives).

4. [F1-score](https://www.google.com/search?sca_esv=e29194d70e62ae1b&cs=1&sxsrf=AE3TifNyIrXo0Yimy163H60bS-aattw3rw%3A1753953347619&q=F1-score&sa=X&ved=2ahUKEwiUmc6x4eaOAxUo3TgGHSB3GToQxccNegQIVBAC&mstk=AUtExfBT-Sivlx_STngzLFUKWRpeNvRYoLie-6HN6jb3Xbzu7nHQimdBV81gKTFQ--S46rbX2wIr-oMuNtb2Bm_1t7ki626LG1I3ezhOrFJKtd0ESeZoJi3XoxG4zoEtiniyIFwxypJwOrk03FzTa3UWYwwSng0QN41x8OMkO60kKycpGXI&csui=3): The [harmonic mean](https://www.google.com/search?sca_esv=e29194d70e62ae1b&cs=1&sxsrf=AE3TifNyIrXo0Yimy163H60bS-aattw3rw%3A1753953347619&q=harmonic+mean&sa=X&ved=2ahUKEwiUmc6x4eaOAxUo3TgGHSB3GToQxccNegQIVxAB&mstk=AUtExfBT-Sivlx_STngzLFUKWRpeNvRYoLie-6HN6jb3Xbzu7nHQimdBV81gKTFQ--S46rbX2wIr-oMuNtb2Bm_1t7ki626LG1I3ezhOrFJKtd0ESeZoJi3XoxG4zoEtiniyIFwxypJwOrk03FzTa3UWYwwSng0QN41x8OMkO60kKycpGXI&csui=3) of precision and recall. It balances both metrics, providing a more robust measure than accuracy, especially with imbalanced datasets.

Formula: 2 \* (Precision \* Recall) / (Precision + Recall)

Example**:** A good F1-score indicates a balance between precision and recall, meaning the model is neither overly cautious nor overly eager to predict the positive class.

5). [AUC-ROC](https://www.google.com/search?sca_esv=e29194d70e62ae1b&cs=1&sxsrf=AE3TifNyIrXo0Yimy163H60bS-aattw3rw%3A1753953347619&q=AUC-ROC&sa=X&ved=2ahUKEwiUmc6x4eaOAxUo3TgGHSB3GToQxccNegQIcRAC&mstk=AUtExfBT-Sivlx_STngzLFUKWRpeNvRYoLie-6HN6jb3Xbzu7nHQimdBV81gKTFQ--S46rbX2wIr-oMuNtb2Bm_1t7ki626LG1I3ezhOrFJKtd0ESeZoJi3XoxG4zoEtiniyIFwxypJwOrk03FzTa3UWYwwSng0QN41x8OMkO60kKycpGXI&csui=3) (Area Under the Receiver Operating Characteristic curve): A plot of the True Positive Rate (recall) against the False Positive Rate at various threshold settings. The area under this curve (AUC) is a single number representing overall performance.

Example:

An AUC of 0.8 means the model has an 80% chance of correctly classifying a randomly chosen positive instance before a randomly chosen negative instance.

These Metrics Are Important as:

* Help compare models objectively.
* Reflect real-world cost of false positives/negatives.
* Guide threshold tuning and model calibration.
* Necessary for regulatory and business reporting.

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:

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

# Step 1: Load dataset

data = load\_breast\_cancer()

df = pd.DataFrame(data.data, columns=data.feature\_names)

df['target'] = data.target

# Step 2: Split into features and target

X = df.drop('target', axis=1)

y = df['target']

# Step 3: Train/test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 4: Train Logistic Regression model

model = LogisticRegression(max\_iter=10000)

model.fit(X\_train, y\_train)

# Step 5: Predict and evaluate

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print("Test Set Accuracy:", accuracy)

**Output:** Test Accuracy: 0.956140350877193

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:

import pandas as pd

from sklearn.datasets import load\_diabetes

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

# Step 1: Load the dataset

diabetes = load\_diabetes()

X = pd.DataFrame(diabetes.data, columns=diabetes.feature\_names)

y = pd.Series(diabetes.target)

# Step 2: Convert regression target to binary classification (above/below median)

y\_binary = (y > y.median()).astype(int)

# Step 3: Split into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_binary, test\_size=0.2, random\_state=42)

# Step 4: Train Logistic Regression with L2 regularization

model = LogisticRegression(penalty='l2', solver='liblinear')

model.fit(X\_train, y\_train)

# Step 5: Print coefficients

print("Model Coefficients:")

for feature, coef in zip(X.columns, model.coef\_[0]):

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

# Step 6: Evaluate the model

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print("\nTest Set Accuracy:", accuracy)

**Output:**

Model Coefficients:

age: 0.6918

sex: -0.6866

bmi: 2.6622

bp: 2.0649

s1: 0.4081

s2: 0.1739

s3: -1.6855

s4: 1.5280

s5: 2.4391

s6: 1.4289

Test Set Accuracy: 0.7415730337078652

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:

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

# Step 1: Load the dataset

iris = load\_iris()

X = pd.DataFrame(iris.data, columns=iris.feature\_names)

y = pd.Series(iris.target)

# Step 2: Split into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 3: Train Logistic Regression with multi\_class='ovr'

model = LogisticRegression(multi\_class='ovr', solver='liblinear')

model.fit(X\_train, y\_train)

# Step 4: Predict and evaluate

y\_pred = model.predict(X\_test)

print("Classification Report:\n")

print(classification\_report(y\_test, y\_pred, target\_names=iris.target\_names))

**Output:**

**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\_digits

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import GridSearchCV, train\_test\_split

from sklearn.metrics import accuracy\_score

# Step 1: Load digits dataset

digits = load\_digits()

X = digits.data

y = digits.target

# Step 2: Split data (optional, but we can use full data for GridSearchCV)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 3: Define Logistic Regression and parameter grid

logreg = LogisticRegression(max\_iter=10000, solver='liblinear')

param\_grid = {

'C': [0.01, 0.1, 1, 10],

'penalty': ['l1', 'l2']

}

# Step 4: Grid Search with cross-validation

grid = GridSearchCV(logreg, param\_grid, cv=5, scoring='accuracy')

grid.fit(X\_train, y\_train)

# Step 5: Print best parameters and validation score

print("Best Parameters:", grid.best\_params\_)

print("Best Cross-Validation Accuracy:", grid.best\_score\_)

# Step 6: Evaluate on test set

y\_pred = grid.predict(X\_test)

test\_accuracy = accuracy\_score(y\_test, y\_pred)

print("Test Set Accuracy:", test\_accuracy)

**Output:**

Best Parameters: {'C': 0.1, 'penalty': 'l1'}

Best Cross-Validation Accuracy: 0.9672861014324429

Test Set Accuracy: 0.9611111111111111

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:

from sklearn.datasets import load\_wine

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

wine = load\_wine()

X, y = wine.data, wine.target

# Split into train and test sets

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\_scaling = LogisticRegression(max\_iter=10000)

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)

# Standardize the features

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Logistic Regression with scaling

model\_with\_scaling = LogisticRegression(max\_iter=10000)

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 the results

print("Accuracy without scaling: {:.4f}".format(accuracy\_no\_scaling))

print("Accuracy with scaling : {:.4f}".format(accuracy\_with\_scaling))

**Output:**

**Accuracy without scaling: 1.0000**

**Accuracy with scaling: 1.0000**

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. Understand the Business Goal**

* **Objective**: Predict whether a customer will respond to a marketing campaign.
* **Challenge**: Highly **imbalanced data** (only 5% positives).
* **Success Metric**: Not overall accuracy — focus on **precision**, **recall**, or **F1-score** for the **positive class (responders)**.

**2. Data Preparation**

**a. Explore & Clean Data**

* Handle missing values.
* Encode categorical features (OneHotEncoder or LabelEncoder).
* Remove duplicates/outliers if they don't reflect real customer behavior.

**b. Feature Engineering**

* Use domain knowledge to:
  + Derive features like *recency of purchase*, *average basket size*, *engagement score*, etc.
  + Normalize date fields (e.g., days since last purchase).

**3. Handle Imbalanced Classes**

Use one or a combination of the following:

**a. Resampling**

* **Oversample** the minority class (e.g., SMOTE).
* **Undersample** the majority class (e.g., RandomUnderSampler).

**b. Class Weights**

* Use LogisticRegression(class\_weight='balanced') to penalize misclassification of the minority class.

**4. Feature Scaling:**

**Standardize features using StandardScaler:**

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

**5. Train/Test Split**

from sklearn.model\_selection import train\_test\_spilt

X\_train, X\_test, y\_train, y\_test = train\_test\_spilt(X\_scaled, y, stratify=y, test\_size=0.2, random\_state=42)

**6. Model Training with Hyperparameter Tuning:**

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import GridSearchCV

param\_grid = {

‘C’: [0.01, 0.1, 1, 10]

‘penalty’: [‘11’, ‘12’],

‘solver’: [‘liblinear’],

}

lr = LogisticsRegression(class\_weight=’balanced’, max\_iter=1000)

grid = GridSearchCV(lr, param\_grid, scoring=’f1’, cv=5)

grid.fit(X\_train, y\_train)

**7. Model Evaluation:**

Since the data is imbalaced, avoid accuracy.Use:

**.** Confusion Matrix

**.** Precision, Recall, F1-Score

**.** ROC AUC

**.** Precision-Recall Curve (better for imbalanced data)

from sklearn.metrics import classification\_report, confusion\_matrix, roc\_auc\_score, precision\_recall\_curve

y\_pred = grid.predict(X\_test)

print(classification\_report(y\_test, y\_pred, digits=4))

**8. Threshold Tuning:**

Default threshold of 0.5 may not be optimal.

. Use the Precision-Recall curve to find a threshold that balances false positives and false negatives for your business goals(e.g., targeting high-probability customers only).

**9. Deployment and Monitoring:**

. Deploy model with probability output.

. Log predictions and actual response.

. Retrain regularly as customer behaviour changes.