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

Ans:-Logistic Regression is a machine learning algorithm used to predict categories, especially when there are only two classes (called binary classification).

- For example:
  i.Will a customer buy a product? (Yes or No)
  ii.Is an email spam or not spam?

Instead of predicting a number (like Linear Regression), it predicts a probability — a number between 0 and 1 — which is then used to decide the class (e.g., if the probability is > 0.5, classify as "Yes").

It uses a sigmoid function to make sure the output is always between 0 and 1

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

Ans:-The sigmoid function is a mathematical function that maps any real-valued number to a value between 0 and 1.

Role in Logistic Regression

In Logistic Regression, the sigmoid function is used to:

1. Convert Linear Output into a Probability

i.The model first calculates a linear combination of input features (just like in linear regression).
ii.But this linear output can be any real number — and we need a probability (between 0 and 1).
iii.The sigmoid function converts this number into a probability.

2. Enable Classification

Once we have the probability, we can classify the input:

i.If σ(z)>0.5\sigma(z) > 0.5σ(z)>0.5, classify as class 1

ii.If σ(z)<0.5\sigma(z) < 0.5σ(z)<0.5, classify as class 0

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

Answer:-Regularization is a technique used in logistic regression (and other models) to prevent overfitting by adding a penalty term to the loss (cost) function.

Regularization Needed

a.)Logistic regression can overfit the training data, especially when:
i.There are too many features
ii.Some features are irrelevant or noisy
b.)Overfitting means the model performs well on training data but poorly on unseen/test data.

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

Answer: Evaluation metrics help us measure how well a classification model performs, especially when it's applied to new, unseen data.

They are important because:

i.Accuracy alone may be misleading, especially with imbalanced data
ii**.**Different problems (e.g., medical diagnosis vs. spam detection) require different metrics
iii.Metrics help in model selection and improvement

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)

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()

X = pd.DataFrame(data.data, columns=data.feature_names)

y = pd.Series(data.target)


# Step 2: Split into train and test 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)


# Step 3: Train Logistic Regression model

model = LogisticRegression(max_iter=10000)  # Increase iterations
if needed

model.fit(X_train, y_train)


# Step 4: Predict and calculate accuracy

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
```

Step 5: Print accuracy

print(f"Logistic Regression Model Accuracy: {accuracy:.2f}")

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 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()

X = pd.DataFrame(data.data, columns=data.feature_names)

y = pd.Series(data.target)

Step 2: Split into train and test 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

)

Step 3: Train Logistic Regression with L2 regularization

model = LogisticRegression(penalty='l2', solver='lbfgs', max_iter=10000)

model.fit(X_train, y_train

```
# Step 4: Predict and evaluate
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
# Step 5: Print model coefficients and accuracy
print("Model Coefficients (L2 Regularization):")
for feature, coef in zip(X.columns, model.coef_[0]):
    print(f"{feature}: {coef:.4f}")


print(f"\nModel Accuracy: {accuracy:.2f}")
```

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)

Answer:-

```
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 = iris.data
y = 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=0.3, random_state=42)

Step 3: Train a Logistic Regression model with One-vs-Rest

model = LogisticRegression(multi_class='ovr', solver='liblinear') # 'liblinear' supports ovr

model.fit(X_train, y_train)

Step 4: Make predictions

y_pred = model.predict(X_test)

Step 5: Print the classification report

print("Classification Report:\n")

print(classification_report(y_test, y_pred, target_names=iris.target_names))

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: from sklearn.datasets import load_iris

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import GridSearchCV, train_test_split

from sklearn.metrics import accuracy_score

Load dataset

iris = load_iris()

X = iris.data

y = iris.target

```
# Split dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Define Logistic Regression model

logreg = LogisticRegression(solver='liblinear', multi_class='ovr')

# Define hyperparameter grid to search

param_grid = {

    'C': [0.01, 0.1, 1, 10, 100],          # Regularization strength

    'penalty': ['l1', 'l2']                # Regularization type

}

# Setup GridSearchCV

grid_search = GridSearchCV(logreg, param_grid, cv=5, scoring='accuracy')

# Fit GridSearchCV

grid_search.fit(X_train, y_train)

# Best parameters

print("Best Parameters:", grid_search.best_params_)

# Predict on the test set using the best estimator

best_model = grid_search.best_estimator_

y_pred = best_model.predict(X_test)

# Validation accuracy on test set

accuracy = accuracy_score(y_test, y_pred)

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

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_iris

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

iris = load_iris()

X = iris.data

y = iris.target

# Split dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Logistic Regression without scaling

model_no_scaling = LogisticRegression(solver='liblinear', multi_class='ovr')

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)

print(f"Accuracy without scaling: {accuracy_no_scaling:.4f}")
```

```
# Standardize 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(solver='liblinear',
multi_class='ovr')
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(f"Accuracy with scaling: {accuracy_with_scaling:.4f}")
```

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:-Approach to Building the Model

1. Understand the Data & Problem

i.Response rate is only 5%, so this is a highly imbalanced classification problem.

ii.Business goal: correctly identify potential responders (positives) to optimize marketing efforts and reduce wasted costs

2. Data Handling

i.Collect relevant features:- customer demographics, past purchase behavior, browsing history, campaign interaction, etc.
ii.Data cleaning:- handle missing values, outliers, and inconsistencies.
iii.Feature engineering:- create meaningful features such as total spend, recency, frequency, average basket size, or engagement scores.

3. Feature Scaling

i.Scale numerical features using StandardScaler or MinMaxScaler because Logistic Regression benefits from standardized inputs.
ii.Avoid data leakage: fit scaler only on training data and apply to test/validation sets.

4. Handling Imbalanced Classes

Since only 5% respond, the model might bias towards predicting the majority class (non-responders).

5. Hyperparameter Tuning

i.Use GridSearchCV or RandomizedSearchCV to tune:
ii.`C` (inverse of regularization strength).
iii.`penalty` (e.g., 'l1' or 'l2').
iv.Solver compatible with penalty.

v.`class_weight` (try 'balanced' or custom weights).
vi.Use stratified cross-validation to maintain class distribution in folds.

6. Model Evaluation

Because of imbalance, accuracy is misleading.

Use metrics that focus on the minority class:

Precision: How many predicted responders are actually responders.
Recall (Sensitivity): How many actual responders are correctly identified.
F1-score: Harmonic mean of precision and recall.
ROC-AUC: Area under the Receiver Operating Characteristic curve.
PR-AUC: Area under the Precision-Recall curve (more informative with imbalanced data).

7. Final Model Deployment & Monitoring

i.Once the model is trained and validated:
ii.Deploy in production to score new customers.
iii.Monitor model performance over time for data drift.
iv.Retrain periodically with fresh data.