ASSIGNMENT SOLUTION

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. It predicts the probability that a given input belongs to a particular category—typically binary (e.g., yes/no, 0/1, spam/not spam).

It uses the logistic (sigmoid) function to map predicted values to probabilities between 0 and 1.

The output is interpreted as the likelihood of a certain class.

Common use cases: disease diagnosis, email spam detection, customer churn prediction.

Difference #1: Type of Response Variable

A linear regression model is used when the response variable takes on a continuous value such as:

- Price
- Height
- Age
- Distance

Conversely, a logistic regression model is used when the response variable takes on a categorical value such as:

- Yes or No
- Male or Female
- Win or Not Win

Difference #2: Equation Used

Linear regression uses the following equation to summarize the relationship between the predictor variable(s) and the response variable:

Y = β0 + β1X1 + β2X2 + … + βpXp

where:

- Y: The response variable
- Xj: The jth predictor variable
- βj: The average effect on Y of a one unit increase in Xj, holding all other predictors fixed

Conversely, logistic regression uses the following equation:

p(X) = eβ0 + β1X1 + β2X2 + … + βpXp / (1 + eβ0 + β1X1 + β2X2 + … + βpXp)

Difference #3: Method Used to Fit Equation

- Linear regression uses a method known as ordinary least squares to find the best fitting regression equation.
- Conversely, logistic regression uses a method known as maximum likelihood estimation to find the best fitting regression equation.

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

Answer

The sigmoid function, also known as the logistic function, is defined as:

$\sigma(z)$ = 1/ 1 + $e - z$

Here, $z$is a linear combination of input features: z= β 0 + β 1 x 1 + ⋯ + β n x n

The output of σ (z) is always between 0 and 1, making it perfect for representing probabilities.

Role in Logistic Regression

1.Probability Mapping

- Logistic regression calculates a linear score z from input features.
- The sigmoid function converts this score into a probability that the input be longs to the positive class (e.g., “yes”, “spam”, “disease”).

2.Decision Boundary

- If σ (z) > 0.5 , the model predicts class 1.
- If σ (z) < 0.5 , it predicts class 0.
- This threshold can be adjusted depending on the problem.

3.Smooth Gradient for Optimization

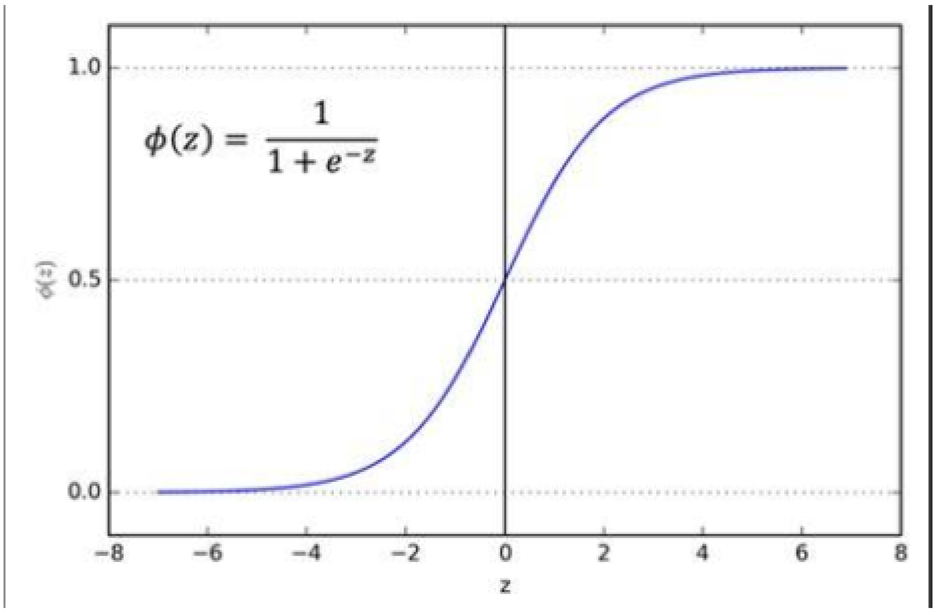
- The sigmoid function is differentiable, which allows gradient-based optimization methods (like gradient descent) to efficiently update model parameters.

4.Interpretability

- The output is intuitive: a value like 0.85 means there's an 85% chance the input belongs to the positive class.

Visual Insight

- The sigmoid curve is S-shaped, smoothly transitioning from 0 to 1:
- For large negative z , output ≈ 0
- For large positive z , output ≈ 1
- At z=0 , output = 0.5 (the tipping point)
- This shape ensures that extreme values don't produce extreme predictions—they're capped within a probability range



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

Answer

Regularization is a technique used to penalize model complexity by adding a constraint to the loss function during training. In logistic regression, this helps ensure the model generalizes well to unseen data.

There are two main types:

L1 Regularization (Lasso): Adds the absolute value of coefficients to the loss function. It can shrink some coefficients to zero, effectively performing feature selection.

L2 Regularization (Ridge): Adds the squared value of coefficients to the loss function. It discourages large weights but doesn’t eliminate them entirely.

1. Ridge Regression (L2 Regularization)

Penalty term: Adds the sum of the squared values of the model coefficients to the cost function. This is known as L2 regularization.

Cost function: [J(\theta) = \text{RSS} + \lambda \sum_{j=1}^{n} \theta_j^2] Where:

(\text{RSS}) is the residual sum of squares (the error term of the model). (\theta_j) are the model parameters. (\lambda) is the regularization parameter controlling the strength of the penalty. Effect on coefficients:

Ridge regression penalizes large coefficients by shrinking them, but it never forces them to zero. It keeps all features in the model, just making their coefficients smaller. This is useful when you believe that most features are relevant and you want to reduce the impact of less important features without discarding them. When to use:

Use Ridge regression when you believe there is multicollinearity (high correlation between features) in your data and want to prevent large coefficients but don't want to eliminate any features entirely.

2. Lasso Regression (L1 Regularization)

Penalty term: Adds the sum of the absolute values of the model coefficients to the cost function. This is known as L1 regularization.

Cost function: [J(\theta) = \text{RSS} + \lambda \sum_{j=1}^{n} |\theta_j|] Where:

(\text{RSS}) is the residual sum of squares (the error term of the model). (\theta_j) are the model parameters. (\lambda) is the regularization parameter controlling the strength of the penalty. Effect on coefficients:

Lasso regression tends to drive some coefficients exactly to zero, effectively performing feature selection. It will completely eliminate less important features by setting their coefficients to zero. This can be very helpful when you believe that some features are irrelevant or redundant, and you want to automatically exclude them from the model. When to use:

Use Lasso regression when you suspect that many features are irrelevant or when you need automatic feature selection.

Why is Regularization Needed in Logistic Regression?

1.Overfitting Prevention:

Overfitting occurs when the model fits the training data too closely, capturing noise and random fluctuations in the data rather than the underlying trend. This results in poor performance on unseen data because the model is too specialized to the training set. Regularization helps by reducing the complexity of the model, making it less likely to overfit, and improving the model's ability to generalize.

2.Control Model Complexity:

A model with many features (predictors) may learn overly complex relationships, making the model more prone to overfitting. Regularization encourages smaller model parameters, which simplifies the model and forces it to focus on the most important features, preventing it from relying too heavily on individual features that may just be noise.

3.Prevents Large Weights:

Without regularization, the parameters of the model ((\theta)) can grow very large if the training data is noisy or the model is overly complex. Large weights can lead to unstable predictions, especially for new data. Regularization keeps the weights smaller, making the model more stable and robust. Explain the difference between Lasso, Ridge, and Elastic Net regression? Lasso, Ridge, and Elastic Net regression are all variations of regularized linear regression methods that add penalty terms to the cost function in order to prevent overfitting and improve generalization. They differ in the way the regularization term is applied to the coefficients (parameters) of the model.

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

Answer

1. Accuracy

- Definition: The proportion of correctly classified instances out of all instances in the dataset.

Formula: [\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}] where:

- TP = True Positives (correctly predicted positive instances)

- TN = True Negatives (correctly predicted negative instances)
- FP = False Positives (incorrectly predicted as positive)
- FN = False Negatives (incorrectly predicted as negative)

Usefulness:

- Simple and widely used. *Best used when the classes are balanced (i.e., both positive and negative classes have roughly the same number of instances).

Limitation:

- Accuracy can be misleading in imbalanced datasets (where one class dominates the other), as it may appear high even if the model is not performing well on the minority class.

2. Precision

- Definition: The proportion of correctly predicted positive instances out of all instances that were predicted as positive.

Formula: [\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}]

Usefulness:

- Important when the cost of false positives (e.g., classifying a negative instance as positive) is high.
- Precision is used in contexts where you want to ensure that when the model predicts a positive class, it's truly positive (e.g., in spam detection).

3. Recall (Sensitivity or True Positive Rate)

- Definition: The proportion of correctly predicted positive instances out of all actual positive instances.

Formula: [\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}]

Usefulness:

- Recall is important when the cost of false negatives (e.g., classifying a positive instance as negative) is high.
- Used in medical testing or fraud detection, where you don't want to miss any true positives (e.g., detecting a disease or fraud).

4. F1 Score

- Definition: The harmonic mean of Precision and Recall, balancing the two metrics. It is useful when you need a balance between Precision and Recall.

Formula: [\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}]

Usefulness:

- The F1 score is a single metric that captures both Precision and Recall, making it useful when the classes are imbalanced and you care about both false positives and false negatives.
- Commonly used when there is a need to balance the trade-off between Precision and Recall.

5. Specificity (True Negative Rate)

- Definition: The proportion of correctly predicted negative instances out of all actual negative instances.

Formula: [\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}]

Usefulness:

- Specificity is useful in scenarios where you want to focus on correctly identifying negative instances, such as in fraud detection where you don't want to mistakenly flag a legitimate transaction as fraudulent.

6. ROC Curve (Receiver Operating Characteristic Curve)

- Definition: A graphical plot that illustrates the diagnostic ability of a binary classifier as its discrimination threshold is varied.

Key Elements:

- True Positive Rate (TPR) or Recall on the Y-axis.
- False Positive Rate (FPR) on the X-axis.
- The curve shows the trade-off between Recall and False Positive Rate at different threshold settings.

Usefulness:

- The ROC curve provides a comprehensive view of the classifier’s performance across different thresholds, helping to evaluate how well the model discriminates between classes.
- The area under the curve (AUC) is a common metric to summarize the ROC curve performance.

7. AUC-ROC (Area Under the ROC Curve)

- Definition: The area under the ROC curve, also known as AUC, represents the probability that a randomly chosen positive instance is ranked higher than a randomly chosen negative instance.

Formula:

- AUC is the area under the ROC curve, which ranges from 0 to 1.

Usefulness:

- AUC is a single value that summarizes the performance of the model. A higher * AUC value indicates a better model.
- AUC is especially useful when comparing models, as it is independent of the * classification threshold and works well in imbalanced datasets.

Interpretation:

- AUC = 0.5: The model performs no better than random chance.
- AUC = 1: The model perfectly classifies the data.

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
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression # Import LogisticRegressior

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

# Split into train/test sets (80/20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train logistic regression model
model = LogisticRegression(random_state=42)
model.fit(X_train_scaled, y_train)

# Predict and calculate accuracy
y_pred = model.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
```

```
# Print accuracy
print(f"Test Accuracy: {accuracy:.4f}")
```

```
Test Accuracy: 0.9737
```

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 necessary libraries
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

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

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

# Train Logistic Regression model using L2 regularization (Ridge)
model = LogisticRegression(penalty='l2', C=1.0, solver='liblinear', max_iter=1

model.fit(X_train, y_train)

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

# Print model coefficients
print("Model Coefficients (L2 Regularization):\n")
for feature, coef in zip(X.columns, model.coef_[0]):
    print(f"{feature}: {coef:.4f}")

# Print accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"\nModel Accuracy: {accuracy:.4f}")
```

```
Model Coefficients (L2 Regularization):

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

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)

Answer

```
# Import 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 classification_report

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

# Split the dataset into training and test sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

# Train a Logistic Regression model for multiclass classification using OvR
model = LogisticRegression(multi_class='ovr', solver='liblinear', max_iter=1000
model.fit(X_train, y_train)

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

# Print the classification report
print("Classification Report:\n")
print(classification_report(y_test, y_pred, target_names=data.target_names))
```

```
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

/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_logistic.py:1256:
  warnings.warn(
```

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)

Answer

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_breast_cancer # Using Breast Cancer dataset
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Load the Breast Cancer dataset from sklearn
```

```
print("Loading Breast Cancer dataset from sklearn...")
cancer = load_breast_cancer()
X, y = cancer.data, cancer.target

print(f"Dataset shape: {X.shape}")
print(f"Classes: {cancer.target_names}")
print(f"Class distribution: {np.bincount(y)}")

# Split into train/test sets (using the same split as Q6 for consistency)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Standardize features (important for regularization)
print("\nStandardizing features...")
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print(f"Training set shape: {X_train_scaled.shape}")
print(f"Test set shape: {X_test_scaled.shape}")


param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2']
}
model = LogisticRegression(solver='liblinear', random_state=42, max_iter=1000)

print("\nPerforming GridSearchCV...")
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scorir

# Fit GridSearchCV on the scaled training data
grid_search.fit(X_train_scaled, y_train)

print("GridSearchCV complete.\n")

# Print the best parameters found
print("Best parameters found by GridSearchCV:")
print(grid_search.best_params_)

# Print the best cross-validation score (accuracy)
print("\nBest cross-validation accuracy:")
print(f"{grid_search.best_score_:.4f}")

# Evaluate the best model on the test set
best_model = grid_search.best_estimator_
y_pred_test = best_model.predict(X_test_scaled)
test_accuracy = accuracy_score(y_test, y_pred_test)

print("\nTest set accuracy with best parameters:")
print(f"{test_accuracy:.4f} ({test_accuracy*100:.2f}%)")

# Optional: Print classification report for the best model on the test set
from sklearn.metrics import classification_report
```

```
print("\nClassification Report on Test Set with Best Model:")
print(classification_report(y_test, y_pred_test, target_names=cancer.target_nam
```

```
Loading Breast Cancer dataset from sklearn...
Dataset shape: (569, 30)
Classes: ['malignant' 'benign']
Class distribution: [212 357]

Standardizing features...
Training set shape: (455, 30)
Test set shape: (114, 30)

Performing GridSearchCV...
GridSearchCV complete.

Best parameters found by GridSearchCV:
{'C': 1, 'penalty': 'l2'}

Best cross-validation accuracy:
0.9802

Test set accuracy with best parameters:
0.9825 (98.25%)

Classification Report on Test Set with Best Model:
              precision    recall  f1-score   support

   malignant       0.98      0.98      0.98        42
      benign       0.99      0.99      0.99        72

    accuracy                           0.98       114
   macro avg       0.98      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

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)

Answer

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_breast_cancer # Using Breast Cancer dataset
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 the Breast Cancer dataset from sklearn
print("Loading Breast Cancer dataset from sklearn...")
cancer = load_breast_cancer()
X, y = cancer.data, cancer.target

print(f"Dataset shape: {X.shape}")
print(f"Classes: {cancer.target_names}")
print(f"Class distribution: {np.bincount(y)}")

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

print(f"\nTraining set shape: {X_train.shape}")
print(f"Test set shape: {X_test.shape}")

# --- Train model WITHOUT scaling ---
print("\nTraining Logistic Regression model WITHOUT scaling...")
model_no_scale = LogisticRegression(random_state=42, max_iter=1000, solver='lib
model_no_scale.fit(X_train, y_train)
y_pred_no_scale = model_no_scale.predict(X_test)
accuracy_no_scale = accuracy_score(y_test, y_pred_no_scale)

print(f"Accuracy WITHOUT scaling: {accuracy_no_scale:.4f} ({accuracy_no_scale*1

# --- Train model WITH scaling ---
print("\nStandardizing features...")
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
print("Training Logistic Regression model WITH scaling...")
model_scaled = LogisticRegression(random_state=42, max_iter=1000, solver='libli
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)

print(f"Accuracy WITH scaling:    {accuracy_scaled:.4f} ({accuracy_scaled*100:.

# --- Comparison ---
print("\nComparison of Accuracy:")
print(f"  Without Scaling: {accuracy_no_scale:.4f}")
print(f"  With Scaling:    {accuracy_scaled:.4f}")
```

```
Loading Breast Cancer dataset from sklearn...
Dataset shape: (569, 30)
Classes: ['malignant' 'benign']
Class distribution: [212 357]

Training set shape: (455, 30)
Test set shape: (114, 30)

Training Logistic Regression model WITHOUT scaling...
Accuracy WITHOUT scaling: 0.9561 (95.61%)
```

```
Standardizing features...
Training Logistic Regression model WITH scaling...
Accuracy WITH scaling:    0.9825 (98.25%)

Comparison of Accuracy:
  Without Scaling: 0.9561
  With Scaling:    0.9825
```

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. Data Handling and Preprocessing:

Data Loading and Exploration: Load the customer data (features like purchase history, demographics, browsing behavior, past campaign interactions, etc.) into a Pandas DataFrame. Perform initial exploratory data analysis (EDA) to understand feature distributions, identify missing values, and analyze the class distribution (confirm the 5% response rate).

- Feature Engineering: Create new features that might be predictive of response. This could include:

Recency, Frequency, Monetary (RFM) values.

Number of visits in the last X days.

Time spent on site.

Category preferences.

Interaction counts with previous campaigns.

- Handling Missing Values: Impute or remove missing values based on their extent and the nature of the feature.
- Encoding Categorical Features: Convert categorical variables (e.g., gender, location) into numerical formats using techniques like One-Hot Encoding.

2. Feature Scaling:

Standardization or Normalization: Since Logistic Regression uses gradient descent and can be sensitive to feature scales, it's crucial to scale the numerical features. StandardScaler (z-score normalization) or MinMaxScaler are common choices. Apply the scaling after splitting the data to prevent data leakage from the test set into the training process.

3. Handling Class Imbalance:

This is a critical step for imbalanced datasets. Directly training on the imbalanced data will likely result in a model that predicts the majority class (non-responders) most of the time, leading to high accuracy but poor performance on the minority class (responders), which is the class of interest. Techniques include:

- Resampling Techniques:
- Oversampling the Minority Class: Duplicate instances of the minority class (responders) to increase their representation. SMOTE (Synthetic Minority Oversampling Technique) is a popular method that creates synthetic samples of the minority class.
- Undersampling the Majority Class: Randomly remove instances of the majority class (non-responders). This can lead to loss of valuable information. Combination Approaches: Techniques like SMOTEENN or SMOTETomek combine oversampling and undersampling.
- Using Class Weights: Logistic Regression models in libraries like scikit-learn allow assigning higher weights to the minority class during training. This tells the model to penalize misclassifications of the minority class more heavily. This is often simpler and performs well compared to resampling.

4. Model Training:

Splitting Data: Split the preprocessed and potentially balanced data into training, validation (optional but recommended), and testing sets. A common split is 70/15/15 or 80/20 for train/test, with a portion of the training data used for validation during hyperparameter tuning. Ensure the split is stratified to maintain the class distribution in each set.

Logistic Regression Model: Initialize a LogisticRegression model from scikit-learn.

5. Hyperparameter Tuning:

Parameters to Tune: Key hyperparameters for Logistic Regression include: C: The inverse of regularization strength. Smaller values mean stronger regularization (L2 by default). This helps prevent overfitting. penalty: 'l1' or 'l2' regularization. L1 can lead to sparser coefficients (feature selection), while L2 shrinks coefficients. solver: The algorithm to use for optimization (e.g., 'liblinear', 'lbfgs', 'saga').

class_weight: Use 'balanced' to automatically adjust weights inversely proportional to class frequencies, or provide a dictionary of weights. Tuning Method: Use techniques like GridSearchCV or RandomizedSearchCV with cross-validation on the training (or training + validation) data to find the optimal combination of hyperparameters that maximizes a suitable evaluation metric.