Logistic Regression

1. Brief Idea

Logistic regression models the probability of a binary outcome by applying the logistic (sigmoid) function to a linear combination of input features. It outputs values between 0 and 1, interpretable as class probabilities. Parameter estimation is performed via maximum likelihood, yielding coefficients that are easy to interpret. It's a go-to baseline for classification tasks when interpretability matters.

2. Context and Backstory

- Originating in statistics in the 1950s for binary response modeling, logistic regression has been widely used in medical diagnosis (e.g., disease vs. no disease) and social sciences survey analysis.
- In modern machine learning, it remains a competitive baseline in Kaggle competitions and is often deployed in production for its simplicity and robustness.
- Its coefficients provide insight into feature importance, making it popular in regulated industries.

3. Mathematical Derivations and Formulae

We model the probability as:

$$P(y=1 \mid x;\theta) = \sigma(\theta^\top x), \quad \sigma(z) = \frac{1}{1+e^{-z}}$$

The log-likelihood for N i.i.d. samples $\{(x^{(i)}, y^{(i)})\}_{i=1}^N$ is:

$$\ell(\theta) = \sum_{i=1}^N \Big[y^{(i)} \log \sigma(\theta^\top x^{(i)}) + (1-y^{(i)}) \log(1-\sigma(\theta^\top x^{(i)}))\Big]$$

Taking the gradient:

$$\nabla_\theta \ell(\theta) = \sum_{i=1}^N \big(y^{(i)} - \sigma(\theta^\top x^{(i)})\big)\, x^{(i)}$$

There is no closed-form solution, so we solve iteratively (e.g., using gradient ascent or Newton–Raphson).

4. Diagrams and Visual Aids

Below is a visualization of the sigmoid function mapping real-valued inputs to the [0,1] interval.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(z):
    return 1 / (1 + np.exp(-z))

z = np.linspace(-10, 10, 200)
plt.plot(z, sigmoid(z))
plt.title('Sigmoid Function')
plt.xlabel('z')
```

```
plt.ylabel('σ(z)')
plt.grid(True)
plt.show()
```

5. Python Implementation

We will generate synthetic data, fit a logistic regression model using scikit-learn, evaluate accuracy, and plot the ROC curve.

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, roc_curve, auc

# Generate synthetic data
X, y = make_classification(n_samples=300, n_features=3, class_sep=1.5, random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

# Fit logistic regression
model = LogisticRegression()
model.fit(X_train, y_train)

# Evaluate accuracy
preds = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, preds))

# Plot ROC curve
probs = model.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, probs)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.2f}")
plt.plot([0, 1], [0, 1], '--', label='Chance')
plt.xlabel("False Positive Rate (FPR)")
plt.ylabel("True Positive Rate (TPR)")
plt.title("ROC Curve")
plt.legend()
plt.show()
```

6. Frequently Asked Questions (FAQ)

Why not use linear regression for classification?
Linear regression can predict values outside [0, 1] and assumes Gaussian noise, making it unsuitable for modeling probabilities.

How to handle multiclass cases?
Use one-vs-rest (OvR) or multinomial logistic regression (softmax).

What if features aren't linearly separable?
Apply feature engineering, add polynomial terms, or use kernel methods.

Why regularize?
To prevent overfitting, control model complexity, and improve generalization.

7. Higher-Order Thinking Questions (HOTS)

1. How would you derive and implement the Newton–Raphson (IRLS) update for logistic regression?
2. In what scenarios might you prefer L1 over L2 regularization, and how does it impact feature selection?
3. How could you extend logistic regression to output calibrated probabilities in highly imbalanced settings?

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt

# Generate synthetic data for binary classification
np.random.seed(0)
N = 200
X = np.random.randn(N, 2)
true_theta = np.array([1.5, -2.0, 0.5])  # bias, w1, w2
z = true_theta[0] + true_theta[1]*X[:,0] + true_theta[2]*X[:,1]
probs = 1 / (1 + np.exp(-z))
y = (probs > 0.5).astype(int)

# Add bias term
X_b = np.c_[np.ones((N, 1)), X]

# Sigmoid function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Logistic regression via gradient ascent
theta = np.zeros(X_b.shape[1])
alpha = 0.1
n_iter = 1000
for i in range(n_iter):
    z = X_b @ theta
    h = sigmoid(z)
    grad = X_b.T @ (y - h) / N
    theta += alpha * grad
    if i % 200 == 0:
        ll = np.mean(y * np.log(h + 1e-8) + (1 - y) * np.log(1 - h + 1e-8))
        print(f"Iter {i}, Log-Likelihood: {ll:.4f}")

print('Estimated coefficients:', theta)

# Plot decision boundary
plt.scatter(X[:,0], X[:,1], c=y, cmap='bwr', alpha=0.5, label='Data')
x1 = np.linspace(X[:,0].min(), X[:,0].max(), 100)
x2 = -(theta[0] + theta[1]*x1) / theta[2]
plt.plot(x1, x2, color='black', label='Decision Boundary')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Logistic Regression (from scratch)')
plt.legend()
plt.show()
```