Logistic Regression – Complete Beginner-to-Advanced Revision Guide

---

1. Introduction

What is Logistic Regression?

Logistic Regression is a **supervised learning** algorithm used for **classification**, primarily **binary classification** (two possible outcomes).

Despite its name, logistic regression is **not a regression model for continuous outputs**. It models the **probability** that an input belongs to a particular class.

Typical output: $P(y = 1 \mid x) \in [0, 1]$

When to Use Logistic Regression

- Binary outcomes (Yes/No, Spam/Not spam)
- Multiclass problems (with extensions)
- Interpretable models
- Probabilistic predictions required

Real-World Applications

- Medical diagnosis (disease vs no disease)
- Credit scoring (default vs non-default)
- Marketing (customer churn)
- Fraud detection
- Admission prediction

---

2. Mathematical Foundations

2.1 Linear Model

We start with a **linear combination** of features:

$$ z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n = \mathbf{w}^T \mathbf{x} $$

Where:

- $\mathbf{w}$ = weights (parameters)
- $\mathbf{x}$ = input features
- z = logit (raw score)

2.2 Why We Need the Sigmoid Function

Linear regression outputs values from $(-\infty, +\infty)$, but probabilities must lie in $[0, 1]$.

Sigmoid Function

$$ \sigma(z) = \frac{1}{1 + e^{-z}} $$

Properties:

- Smooth, differentiable
- Maps real numbers → probabilities
- S-shaped curve

2.3 Logistic Regression Model

Combining linear model and sigmoid:

$$ P(y = 1 \mid x) = \sigma(\mathbf{w}^T \mathbf{x}) $$

2.4 Odds and Log-Odds

Odds

$$ \text{Odds} = \frac{P(y=1)}{1 - P(y=1)} $$

Log-Odds (Logit)

$$ \log \left(\frac{P(y=1)}{1 - P(y=1)} \right) = \mathbf{w}^T \mathbf{x} $$

Key insight: Logistic regression is **linear in log-odds**, not probabilities.

---

3. Loss Function and Training

3.1 Likelihood Function

For a dataset of m samples:

$$ L(\mathbf{w}) = \prod_{i=1}^m P(y_i \mid x_i) $$

For binary labels:

$$ P(y_i \mid x_i) = \hat{y}_i^{y_i} (1 - \hat{y}_i)^{1 - y_i} $$

3.2 Log-Likelihood

Taking log (numerical stability):

$$ \ell(\mathbf{w}) = \sum_{i=1}^m \left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right] $$

3.3 Binary Cross-Entropy Loss

Loss to minimize:

$$ J(\mathbf{w}) = -\frac{1}{m} \sum_{i=1}^m \left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right] $$

3.4 Gradient Descent

Gradient

$$ \frac{\partial J}{\partial w_j} = \frac{1}{m} \sum_{i=1}^m (\hat{y}_i - y_i) x $$

Update Rule

$$ w_j := w_j - \alpha \frac{\partial J}{\partial w_j} $$

Where:

- α = learning rate

---

4. Regularization

4.1 Why Regularization?

Prevents:

- Overfitting
- Large coefficients
- Poor generalization

4.2 L2 Regularization (Ridge)

$$ J(\mathbf{w}) = \text{Loss} + \frac{\lambda}{2m} \sum w_j^2 $$

4.3 L1 Regularization (Lasso)

$$ J(\mathbf{w}) = \text{Loss} + \frac{\lambda}{m} \sum |w_j| $$

Key difference:

- L1 → feature selection (sparse)
- L2 → weight shrinkage

---

5. Model Evaluation

5.1 Confusion Matrix

Actual \ Predicted	0	1
0	TN	FP
1	FN	TP

5.2 Metrics

Accuracy

$$ \frac{TP + TN}{TP + TN + FP + FN} $$

Precision

$$ \frac{TP}{TP + FP} $$

Recall (Sensitivity)

$$ \frac{TP}{TP + FN} $$

F1-Score

$$ 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} $$

5.3 ROC Curve and AUC

- ROC: TPR vs FPR
- AUC measures ranking ability
- Threshold-independent

---

6. Interpretation of Coefficients

Coefficient Meaning

$$ \beta_j > 0 \Rightarrow x_j \text{ increases probability} $$

Odds Ratio

$$ e^{\beta_j} $$

-

1 increases odds - <1 decreases odds

---

7. Assumptions

- Binary outcome
- Independent observations
- Linearity in log-odds
- No multicollinearity
- Large sample size preferred

---

8. Common Pitfalls & Misconceptions

- ❌Interpreting coefficients as linear change in probability
- ❌Using accuracy on imbalanced data
- ❌Forgetting feature scaling
- ❌Assuming normality of variables
- ❌Perfect separation causing infinite coefficients

---

9. Python Implementation

9.1 From Scratch (NumPy)

```python
import numpy as np

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

def train_logistic(X, y, lr=0.1, epochs=1000):
    m, n = X.shape
    w = np.zeros(n)

    for _ in range(epochs):
        z = X @ w
        y_hat = sigmoid(z)
        gradient = (1/m) * X.T @ (y_hat - y)
        w -= lr * gradient

    return w
```

9.2 Using scikit-learn

```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

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

model = LogisticRegression(penalty='l2', solver='lbfgs')
model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

9.3 Decision Boundary Visualization

```
import matplotlib.pyplot as plt

plt.scatter(X[:,0], X[:,1], c=y)
plt.show()
```

---

10. Extensions

- Multinomial Logistic Regression (Softmax)
- One-vs-Rest
- Regularized Logistic Regression
- Bayesian Logistic Regression

---

11. When NOT to Use Logistic Regression

- Highly non-linear boundaries
- Complex feature interactions
- Very small datasets

---

12. Best Practices

- Scale features
- Tune regularization
- Use ROC-AUC for imbalance
- Check multicollinearity (VIF)
- Interpret coefficients carefully

---

13. Summary

Logistic Regression is:

- Simple
- Powerful

- Interpretable
- Foundational for ML understanding

Mastering it builds intuition for:

- Neural networks
- Generalized linear models
- Probabilistic ML

End of Revision Guide