Supervised Learning: Logistic Regression

Binary and Multiclass Classification

Sarwan Ali

Department of Computer Science
Georgia State University

Classification with Logistic Regression

Today's Learning Journey

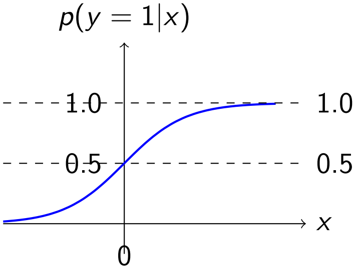
What is Logistic Regression?

Key Characteristics:

- **Classification algorithm** (not regression!)
- Predicts probability of class membership
- Uses sigmoid function for mapping
- Linear decision boundary
- Probabilistic interpretation

Why "Logistic"?

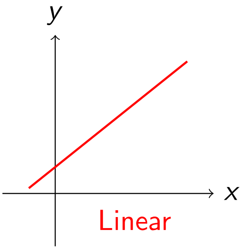
- Uses logistic (sigmoid) function
- Models log-odds (logit) linearly



Linear vs Logistic Regression

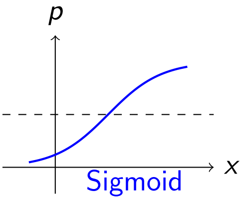
Linear Regression

- Continuous output
- $y = \beta_0 + \beta_1 x$
- Range: $(-\infty, +\infty)$
- Direct prediction



Logistic Regression

- Probability output
- $p = \frac{1}{1+e^{-(\beta_0+\beta_1 x)}}$
- Range: $[0, 1]$
- Probabilistic prediction



The Sigmoid Function

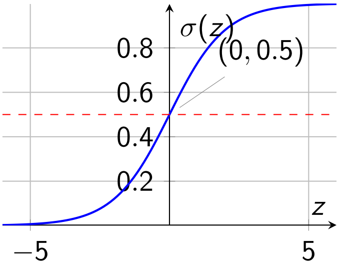
Mathematical Definition:

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

where $z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$

Key Properties:

- Range: $(0, 1)$
- S-shaped curve
- Smooth and differentiable
- $\sigma(0) = 0.5$
- $\sigma(-z) = 1 - \sigma(z)$



As $z \to +\infty$, $\sigma(z) \to 1$

As $z \to -\infty$, $\sigma(z) \to 0$

Why Sigmoid Function?

Advantages of Sigmoid:

1. **Probability Interpretation**
   - Output range $[0, 1]$ represents probabilities
   - Natural threshold at 0.5
2. **Smooth Transition**
   - Continuous and differentiable
   - Smooth decision boundary
3. **Mathematical Properties**
   - Nice derivative: $\sigma'(z) = \sigma(z)(1 - \sigma(z))$
   - Enables gradient-based optimization

Decision Rule:

$$\hat{y} = \begin{cases} 1 & \text{if } \sigma(z) \geq 0.5 \\ 0 & \text{if } \sigma(z) < 0.5 \end{cases}$$

Since $\sigma(z) \geq 0.5$ when $z \geq 0$:

$$\hat{y} = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Binary Logistic Regression Model

Problem Setup:

- Target variable $y \in \{0, 1\}$ (binary)
- Features $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$
- Parameters $\beta = [\beta_0, \beta_1, \ldots, \beta_n]^T$

Model Equations:

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n = \beta^T \mathbf{x} \tag{1}$$

$$p(y = 1|\mathbf{x}) = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{2}$$

$$p(y = 0|\mathbf{x}) = 1 - p(y = 1|\mathbf{x}) = 1 - \sigma(z) \tag{3}$$

Odds and Log-Odds:

$$\text{Odds} = \frac{p(y = 1|\mathbf{x})}{p(y = 0|\mathbf{x})} = \frac{\sigma(z)}{1 - \sigma(z)} = e^z \tag{4}$$

$$\text{Log-Odds (Logit)} = \ln(\text{Odds}) = z = \beta^T \mathbf{x} \tag{5}$$

Geometric Interpretation

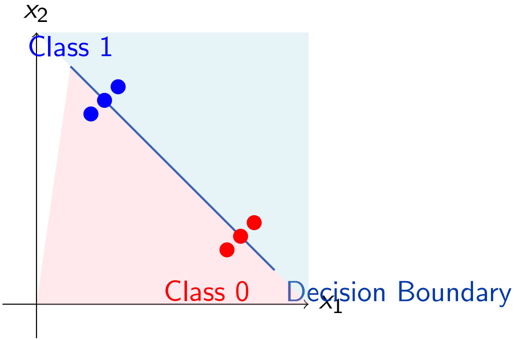
Decision Boundary:

- Linear in feature space
- Defined by $z = 0$
- $\beta^T \mathbf{x} = 0$

For 2D case:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0$$

$$x_2 = -\frac{\beta_0}{\beta_2} - \frac{\beta_1}{\beta_2} x_1$$



Distance from boundary determines confidence

Maximum Likelihood Principle

Goal: Find parameters β that maximize the likelihood of observed data

Given training data: $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m)\}$

Likelihood for one sample:

$$p(y_i|\mathbf{x}_i, \beta) = \sigma(\beta^T \mathbf{x}_i)^{y_i} \cdot (1 - \sigma(\beta^T \mathbf{x}_i))^{1-y_i}$$

Total Likelihood (assuming independence):

$$L(\beta) = \prod_{i=1}^{m} p(y_i|\mathbf{x}_i, \beta) = \prod_{i=1}^{m} \sigma(\beta^T \mathbf{x}_i)^{y_i} \cdot (1 - \sigma(\beta^T \mathbf{x}_i))^{1-y_i}$$

Log-Likelihood:

$$\ell(\beta) = \ln L(\beta) = \sum_{i=1}^{m} \left[y_i \ln(\sigma(\beta^T \mathbf{x}_i)) + (1 - y_i) \ln(1 - \sigma(\beta^T \mathbf{x}_i)) \right]$$

Cost Function and Optimization

From Log-Likelihood to Cost Function:

We want to **maximize** log-likelihood $\ell(\beta)$

Equivalently, **minimize** negative log-likelihood:

$$J(\beta) = -\frac{1}{m}\ell(\beta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y_i \ln(\sigma(\beta^T \mathbf{x}_i)) + (1 - y_i)\ln(1 - \sigma(\beta^T \mathbf{x}_i))\right]$$

This is the Cross-Entropy Loss!

Gradient of Cost Function:

$$\frac{\partial J}{\partial \beta_j} = \frac{1}{m}\sum_{i=1}^{m}(\sigma(\beta^T \mathbf{x}_i) - y_i)x_{ij}$$

Vector Form:

$$\nabla J = \frac{1}{m}\mathbf{X}^T(\sigma - \mathbf{y})$$

where $\sigma = [\sigma(\beta^T \mathbf{x}_1), \ldots, \sigma(\beta^T \mathbf{x}_m)]^T$

Gradient Descent for Logistic Regression

Algorithm:

1. **Initialize:** $\beta^{(0)}$ (usually zeros or small random values)
2. **Repeat until convergence:**

$$z_i^{(t)} = (\beta^{(t)})^T \mathbf{x}_i \tag{6}$$

$$\sigma_i^{(t)} = \frac{1}{1 + e^{-z_i^{(t)}}} \tag{7}$$

$$\beta^{(t+1)} = \beta^{(t)} - \alpha \nabla J(\beta^{(t)}) \tag{8}$$

$$= \beta^{(t)} - \frac{\alpha}{m} \mathbf{X}^T (\sigma^{(t)} - \mathbf{y}) \tag{9}$$

Key Points:

- α is the learning rate
- No closed-form solution (unlike linear regression)
- Cost function is convex $\Rightarrow$ global minimum guaranteed
- Other optimizers: Newton-Raphson, L-BFGS, etc.

Extending to Multiclass Problems

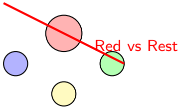
Problem: What if we have $K > 2$ classes?

Two Main Approaches:

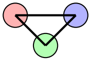
1. **One-vs-Rest (OvR)**
   - Train K binary classifiers
   - Class k vs all other classes
   - Prediction: class with highest probability

2. **One-vs-One (OvO)**
   - Train $\binom{K}{2}$ binary classifiers
   - Every pair of classes
   - Prediction: majority voting

One-vs-Rest



One-vs-One



Multinomial Logistic Regression (Softmax)

Direct Multiclass Extension:

For K classes, we have K sets of parameters: $\beta_1, \beta_2, \ldots, \beta_K$

Softmax Function:

$$p(y = k|\mathbf{x}) = \frac{e^{\beta_k^T \mathbf{x}}}{\sum_{j=1}^{K} e^{\beta_j^T \mathbf{x}}}$$

Properties:

- $\sum_{k=1}^{K} p(y = k|\mathbf{x}) = 1$ (probabilities sum to 1)
- Generalizes sigmoid to multiple classes
- When $K = 2$, reduces to binary logistic regression

Decision Rule:

$$\hat{y} = \arg\max_k p(y = k|\mathbf{x})$$

Softmax: Mathematical Details

Cross-Entropy Loss for Multiclass:

For one-hot encoded labels $\mathbf{y}_i = [y_{i1}, y_{i2}, \ldots, y_{iK}]^T$:

$$J(\beta_1, \ldots, \beta_K) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_{ik} \ln(p(y = k|\mathbf{x}_i))$$

Gradient:

$$\frac{\partial J}{\partial \beta_k} = \frac{1}{m}\sum_{i=1}^{m}(p(y = k|\mathbf{x}_i) - y_{ik})\mathbf{x}_i$$

Comparison with Binary Case:

Aspect	Binary	Multiclass
Activation	Sigmoid	Softmax
Parameters	β (one set)	$\beta_1, \ldots, \beta_K$
Output	Single probability	Probability vector

Advantages and Disadvantages

Advantages:

- Probabilistic output
- No assumptions about feature distributions
- Less prone to overfitting
- Computationally efficient
- No hyperparameters to tune
- Interpretable coefficients
- Linear decision boundary

When to Use Logistic Regression:

- Need probabilistic predictions
- Linear separability exists
- Interpretability is important
- Baseline model for comparison
- Large dataset with simple patterns

Disadvantages:

- Assumes linear relationship
- Sensitive to outliers
- Requires large sample sizes
- Can struggle with complex patterns
- Feature scaling important
- May need feature engineering

Implementation Tips

Preprocessing:

- **Feature Scaling:** Standardize features (mean=0, std=1)
- **Handle Missing Values:** Imputation or removal
- **Categorical Variables:** One-hot encoding

Regularization:

- **L1 (Lasso):** $J(\beta) + \lambda \sum_{j=1}^{n} |\beta_j|$
- **L2 (Ridge):** $J(\beta) + \lambda \sum_{j=1}^{n} \beta_j^2$
- **Elastic Net:** Combination of L1 and L2

Model Evaluation:

- Accuracy, Precision, Recall, F1-score
- ROC curve and AUC
- Confusion matrix
- Cross-validation

Key Takeaways

1. **Logistic Regression** is a classification algorithm that uses the sigmoid function to model probabilities
2. **Sigmoid Function** maps any real number to $(0, 1)$, making it perfect for probability estimation
3. **Maximum Likelihood** principle provides a principled way to find optimal parameters
4. **Linear Decision Boundary** separates classes in feature space
5. **Multiclass Extension** can be achieved through One-vs-Rest, One-vs-One, or Softmax
6. **Practical Algorithm** with good interpretability and efficiency

Remember: Logistic regression models the log-odds linearly!

Example: Email Spam Classification

Problem: Classify emails as spam (1) or not spam (0)

Features:

- x_1: Number of exclamation marks
- x_2: Frequency of word "free"
- x_3: Length of email

Model:

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

$$P(\text{spam}) = \frac{1}{1 + e^{-z}}$$

Interpretation:

- $\beta_1 > 0$: More exclamation marks $\rightarrow$ higher spam probability
- $\beta_2 > 0$: Word "free" increases spam probability
- $\beta_3 < 0$: Longer emails are less likely to be spam

Logistic Regression vs Other Classifiers

Algorithm	**Decision Boundary**	**Probabilistic**	**Interpretability**
Logistic Regression	Linear	Yes	High
SVM	Linear/Non-linear	No	Medium
Decision Trees	Non-linear	Yes	High
k-NN	Non-linear	Yes	Low
Neural Networks	Non-linear	Yes	Low

Key Insight: Logistic regression is the go-to choice when you need:

- Linear separability
- Probability estimates
- Model interpretability
- Fast training and prediction

Assumptions and Model Diagnostics

Key Assumptions:

1. **Linear relationship** between logit and features
2. **Independence** of observations
3. **No severe multicollinearity** among features
4. **Large sample size** (rule of thumb: 10+ events per feature)

Diagnostic Checks:

- **Residual plots:** Check linearity assumption
- **VIF (Variance Inflation Factor):** Detect multicollinearity
- **Cook's distance:** Identify influential points
- **Hosmer-Lemeshow test:** Goodness of fit

Warning Signs:

- Complete/quasi-complete separation
- Very large coefficient values
- Wide confidence intervals

Python Implementation

```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, roc_auc_score

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

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

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

# Make predictions
y_pred = model.predict(X_test_scaled)
y_prob = model.predict_proba(X_test_scaled)[:, 1]

# Evaluate
print(classification_report(y_test, y_pred))
print(f"AUC: {roc_auc_score(y_test, y_prob):.3f}")

# Interpret coefficients
feature_names = ['feature1', 'feature2', 'feature3']
for name, coef in zip(feature_names, model.coef_[0]):
    print(f"{name}: {coef:.3f}")
```

Questions & Discussion

Common Questions:

- Why can't we use linear regression for classification?
- When does logistic regression fail?
- How to handle imbalanced datasets?
- What if features are not linearly separable?

Think about:

- Real-world applications in your field
- When you might choose logistic regression over other methods