Logistic Regression: Overfitting & Regularisation — From Sigmoid to Calibrated Classifiers

Hypothesis, log-loss, gradient descent/Newton steps, and practical regularisation with diagnostics

2019-02-01

Table of contents

Logistic regression becomes far more approachable when each ingredient—hypothesis, loss, optimiser, and regularisation—shows how it shapes the final classifier. This walkthrough presents the concepts, supporting equations, and recommended defaults in a concise sequence.

1 Hypothesis and problem setup

The model computes a linear combination of inputs and maps it through a sigmoid to obtain a probability in $(0, 1)$.

Mathematics. With observations $(x^{(i)}, y^{(i)})$, $y \in \{0, 1\}$, $x \in \mathbb{R}^n$:

$$\log \frac{P(y=1 \mid x)}{1 - P(y=1 \mid x)} = \theta^\top x \iff P(y=1 \mid x) = \sigma(\theta^\top x) = \frac{1}{1 + e^{-\theta^\top x}}.$$

Predict class 1 when $P(y{=}1 \mid x) \geq \tau$ (default $\tau = 0.5$; see the calibration section for alternatives).

Including a bias column $x_0 = 1$ ensures the intercept is learned rather than baked into the features, a common point of failure in scratch implementations.

---

2 Likelihood and log-loss

Cross-entropy loss rewards high probability on the correct class and penalises confident misclassifications.

Mathematics. For i.i.d. Bernoulli labels,

$$\mathcal{L}(\theta) = \prod_{i=1}^{m} \sigma(z^{(i)})^{y^{(i)}} \left(1 - \sigma(z^{(i)})\right)^{1 - y^{(i)}}, \qquad z^{(i)} = \theta^\top x^{(i)}.$$

Taking the negative log-likelihood gives the **log-loss / cross-entropy**:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} \log \sigma(z^{(i)}) + (1 - y^{(i)}) \log(1 - \sigma(z^{(i)})) \right).$$

Vectorised with $X \in \mathbb{R}^{m \times (n+1)}$, $p = \sigma(X\theta)$:

$$J(\theta) = -\frac{1}{m} \left(y^\top \log p + (1 - y)^\top \log(1 - p) \right).$$

Comparing average log-loss with simple classification error illustrates how confident mistakes dominate the optimisation signal even when accuracy is unchanged.

---

3 Optimisation choices

First-order methods rely on gradient information alone, whereas Newton's method also uses curvature to accelerate convergence when the Hessian is well behaved.

Mathematics. The gradient is

$$\nabla_\theta J(\theta) = \frac{1}{m} X^\top (p - y).$$

The Hessian for Newton updates is

$$H(\theta) = \frac{1}{m} X^\top R X, \qquad R = \operatorname{diag}\left(p \odot (1 - p)\right).$$

Options. - Batch or mini-batch gradient descent: simple, scalable, depends on a learning rate α. - Stochastic gradient descent: faster per iteration, good for large datasets. - Newton / IRLS: near-quadratic convergence when m and n are modest.

Contrasting these methods on a small dataset makes the trade-off between per-step cost and convergence speed very clear.

---

4 Bias–variance diagnostics

Comparing training and validation curves reveals whether the model is underfitting or overfitting and whether additional data, features, or regularisation are needed.

Guidelines. - **Underfitting (high bias):** training and validation errors stay high—model too simple or overly regularised. - **Overfitting (high variance):** low training error but high validation error—too many features, too little regularisation.

Diagnostics. Plot learning curves (error vs. sample size), validation curves (error vs. λ or C), and inspect confusion matrices on a holdout set to identify whether capacity or regularisation needs adjustment.

---

5 Regularisation choices

L1 regularisation promotes sparsity by driving some coefficients to zero, while L2 regularisation shrinks coefficients smoothly and stabilises correlated features.

Formulas. With $\lambda \geq 0$ and intercept excluded from penalties: - **L2 (Ridge):** $J_\lambda(\theta) = J(\theta) + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$ — smooth shrinkage, good with correlated features. - **L1 (Lasso):** $J_\lambda(\theta) = J(\theta) + \frac{\lambda}{m} \sum_{j=1}^{n} |\theta_j|$ — drives some coefficients to zero. - **Elastic Net:** combine L1 and L2 to balance sparsity and stability.

Scaling matters. Standardise features (mean 0, variance 1) before penalising so one unit doesn't dominate the penalty. *Never* regularise θ_0.

---

6 Thresholds, imbalance, and calibration

Choosing a classification threshold balances false positives and false negatives; recalibrating the threshold aligns the classifier with current operating requirements.

Practices. - Tune decision threshold τ for your cost trade-offs; use ROC or PR curves depending on imbalance. - Address class imbalance via `class_weight="balanced"`, resampling, or different metrics (PR-AUC, F1, recall at precision). - Calibrate probabilities with Platt scaling or isotonic regression if validation data shows poor calibration.

---

7 Reference NumPy implementation

The vectorised trainer below applies L2 regularisation while leaving the intercept unpenalised so the code mirrors textbook equations.

```
import numpy as np

def sigmoid(z):
    # numerically stable sigmoid
    out = np.empty_like(z, dtype=float)
    pos = z >= 0
    neg = ~pos
    out[pos] = 1.0 / (1.0 + np.exp(-z[pos]))
    expz = np.exp(z[neg])
    out[neg] = expz / (1.0 + expz)
    return out

def log_loss(X, y, theta, lam=0.0):
    m = len(y)
    z = X @ theta
    p = sigmoid(z)
    # clamp to avoid log(0)
    eps = 1e-12
    p = np.clip(p, eps, 1 - eps)
    data = -(y @ np.log(p) + (1 - y) @ np.log(1 - p)) / m
    # L2 penalty (skip intercept)
    reg = lam * (theta[1:] @ theta[1:]) / (2 * m)
    return data + reg

def fit_logreg_l2(X, y, alpha=0.1, lam=0.0, epochs=5000, tol=1e-6):
    # Batch gradient descent with L2 regularisation. X must include a bias column.
    m, n = X.shape
    theta = np.zeros(n)
    last = np.inf
    for it in range(epochs):
        p = sigmoid(X @ theta)
        grad = (X.T @ (p - y)) / m
        grad[1:] += (lam / m) * theta[1:]
        theta -= alpha * grad
        if it % 50 == 0:
            J = log_loss(X, y, theta, lam)
            if abs(last - J) < tol:
                break
            last = J
    return theta

# ---- Demo with synthetic data ----
rng = np.random.default_rng(0)
m = 600
X1 = rng.normal([0, 0], [1.0, 1.0], size=(m//2, 2))
X2 = rng.normal([2.0, 2.0], [1.0, 1.0], size=(m//2, 2))
X_no_bias = np.vstack([X1, X2])
y = np.hstack([np.zeros(m//2, dtype=int), np.ones(m//2, dtype=int)])
```

```python
# Add interactions to tempt overfitting
x1, x2 = X_no_bias[:, 0], X_no_bias[:, 1]
Phi = np.column_stack([np.ones(m), x1, x2, x1 * x2, x1**2, x2**2])

# Standardise non-bias columns
mu, sigma = Phi[:, 1:].mean(0), Phi[:, 1:].std(0) + 1e-8
Phi[:, 1:] = (Phi[:, 1:] - mu) / sigma

# Train with and without regularisation
theta_noreg = fit_logreg_l2(Phi, y, alpha=0.3, lam=0.0, epochs=8000)
theta_l2 = fit_logreg_l2(Phi, y, alpha=0.3, lam=1.0, epochs=8000)

def accuracy(X, y, th):
    p = sigmoid(X @ th) >= 0.5
    return (p == y).mean()

print("Acc (no reg):", accuracy(Phi, y, theta_noreg))
print("Acc (L2=1.0):", accuracy(Phi, y, theta_l2))
print("||theta|| (no reg):", np.linalg.norm(theta_noreg[1:]))
print("||theta|| (L2=1.0):", np.linalg.norm(theta_l2[1:]))
```

---

8 Scikit-learn baseline

```python
# pip install scikit-learn
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(Phi[:, 1:], y, test_size=0.3, random_state=0)

# scikit-learn adds the intercept automatically.
clf = LogisticRegression(
    penalty="l2",        # switch to "l1" or "elasticnet" with solver="saga"
    C=1.0,               # smaller C => stronger regularisation
    solver="liblinear",  # "liblinear" OK for small data; "saga" handles l1/elasticnet
    class_weight="balanced",  # handy for imbalance
    max_iter=2000
).fit(X_train, y_train)

print(classification_report(y_test, clf.predict(X_test)))
```

---

9 Tuning and diagnostics

Sweeping C or λ through a range of values shows how the model responds to different regularisation strengths before selecting a setting for production.

Checklist. - Standardise features and reuse the same transform on validation/test splits. - Search over C (or λ) on a log scale; use cross-validation. - Plot learning curves to decide if you need more data or capacity. - Plot validation curves (metric vs. C) to find the sweet spot. - Inspect confusion matrices, ROC, and PR curves to confirm threshold choices.

---

10 Common pitfalls

- Penalising the intercept (don't).
- Skipping feature scaling before regularisation.
- Reporting accuracy on imbalanced data; prefer PR-AUC, F1, or recall at fixed precision.
- Ignoring collinearity; L1 or elastic net can help.
- Leaking information: compute scaling parameters on the training set only.

---

11 Deployment checklist

- ☐ Add a bias term and standardise features.
- ☐ Use log-loss for training; pick an optimiser (GD/SGD/IRLS).
- ☐ Apply L2/L1/elastic net without penalising the intercept.
- ☐ Tune C (or λ) via cross-validation; inspect learning/validation curves.
- ☐ Select decision thresholds aligned with costs; evaluate with PR/ROC; check calibration.
- ☐ Persist the scaler, coefficients, and chosen threshold for reproducible predictions.

---

12 Where to go next

- Derive and implement stochastic gradient descent with momentum or Adam for large datasets.
- Explore Bayesian logistic regression and compare posterior predictive calibration.
- Extend the calibration section by fitting temperature scaling on neural network logits and contrasting it with Platt scaling.