L02: Logistic Regression

Deep Dive: Mathematical Foundations and Implementation

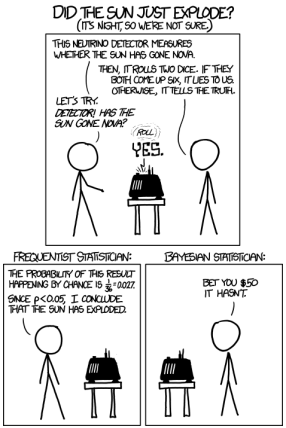
Methods and Algorithms

MSc Data Science

Spring 2026

Outline

The Eternal Debate



XKCD #1132 by Randall Munroe (CC BY-NC 2.5) – "Is the sun going to explode?"

Learning Objectives

By the end of this deep dive, you will be able to:

1. Derive the MLE for logistic regression via gradient of the log-likelihood
2. Analyze model fit using deviance, LRT, AIC/BIC, and Hosmer-Lemeshow
3. Evaluate classification performance using ROC, calibration, and cost-sensitive metrics
4. Apply logistic regression to credit scoring with regulatory interpretation (Basel PD)

Finance Application: Credit scoring and probability of default (PD)

Bloom's Levels 4–5: Analyze, Evaluate, Create

From Linear to Logistic

The Classification Problem

- Given features $\mathbf{x} \in \mathbb{R}^p$, predict $y \in \{0, 1\}$
- Linear regression: $\hat{y} = \mathbf{w}^T\mathbf{x} + b$ produces unbounded output
- Need: $P(y = 1|\mathbf{x}) \in [0, 1]$

Solution: The Logistic Function

$$P(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x}+b)}}$$

The logistic (sigmoid) function maps any real number to the interval $(0, 1)$

The Sigmoid Function

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

Key Properties:

- Range: $(0, 1)$ – perfect for modeling probabilities
- $\sigma(0) = 0.5$ – natural classification threshold
- Symmetric: $\sigma(-z) = 1 - \sigma(z)$

Derivative (crucial for gradient computation):

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

- Maximum at $z = 0$ where $\sigma'(0) = 0.25$
- Vanishes as $|z| \to \infty$ (saturation regions)

The derivative's simple form makes gradient computation elegant

Odds and Log-Odds

The Logit Link Function

- Odds: $\frac{P(y=1)}{P(y=0)} = \frac{p}{1-p}$
- Log-odds (logit): $\log\left(\frac{p}{1-p}\right) = \mathbf{w}^T\mathbf{x} + b$

Coefficient Interpretation via Odds Ratios

- w_j: change in log-odds per unit increase in x_j
- e^{w_j}: **odds ratio** – multiplicative effect on odds
- Example: $w_{\text{income}} = 0.5 \Rightarrow e^{0.5} \approx 1.65$, so each unit increase in income multiplies the odds of approval by 1.65

Odds ratio interpretation is key for regulatory compliance in banking

Maximum Likelihood Estimation

The Likelihood Function
For observations (x_i, y_i), the likelihood is:

$$L(\mathbf{w}) = \prod_{i=1}^{n} p_i^{y_i}(1-p_i)^{1-y_i}$$

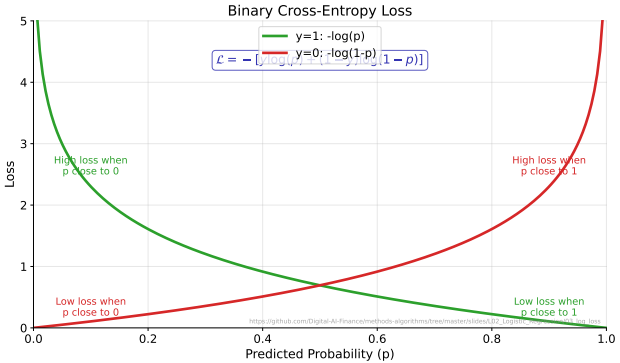
where $p_i = \sigma(\mathbf{w}^T\mathbf{x}_i + b)$

Log-Likelihood (easier to optimize):

$$\ell(\mathbf{w}) = \sum_{i=1}^{n} [y_i \log(p_i) + (1-y_i)\log(1-p_i)]$$

Maximize log-likelihood $\equiv$ minimize negative log-likelihood (cross-entropy loss)

Binary Cross-Entropy Loss



Loss Function:

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(p_i) + (1-y_i)\log(1-p_i)]$$

Cross-entropy is convex in the weights – guaranteed global optimum

Gradient Derivation

Single-Sample Gradient (via chain rule):

$$\frac{\partial \mathcal{L}_i}{\partial w_j} = \underbrace{\frac{\partial \mathcal{L}_i}{\partial p_i}}_{\text{loss w.r.t. pred}} \cdot \underbrace{\frac{\partial p_i}{\partial z_i}}_{\sigma'(z)} \cdot \underbrace{\frac{\partial z_i}{\partial w_j}}_{x_{ij}} = (p_i - y_i)\, x_{ij}$$

Full Gradient in Matrix Form:

$$\nabla_{\mathbf{w}} \mathcal{L} = \frac{1}{n} \mathbf{X}^T (\mathbf{p} - \mathbf{y})$$

where $\mathbf{p} = \sigma(\mathbf{X}\mathbf{w})$

Key Insight: Same form as the linear regression gradient – only $\mathbf{p}$ differs (sigmoid vs. identity).

The elegant gradient form arises from the sigmoid derivative: $\sigma'(z) = \sigma(z)(1 - \sigma(z))$

Gradient Descent Update

Update Rule:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla_{\mathbf{w}} \mathcal{L} = \mathbf{w}^{(t)} - \frac{\eta}{n} \mathbf{X}^T (\sigma(\mathbf{X}\mathbf{w}^{(t)}) - \mathbf{y})$$

Practical Considerations:

- Feature scaling: standardize inputs for faster convergence
- Learning rate: start with $\eta = 0.01$; use line search or decay schedule
- Convergence: monitor loss; stop when $\|\nabla\| < \epsilon$

No closed-form solution (unlike normal equation) – must use iterative optimization

Standard Errors of Coefficients

How Certain Are We About Each Coefficient?
Standard error measures uncertainty in $\hat{w}_j$:

$$\text{SE}(\hat{w}_j) = \sqrt{[\mathbf{H}^{-1}]_{jj}} \tag{1}$$

where $\mathbf{H} = -\mathbf{X}^T \text{diag}\big(\hat{p}_i(1-\hat{p}_i)\big)\mathbf{X}$ is the Hessian of the log-likelihood.

Intuition:

- Small SE $\Rightarrow$ coefficient is precisely estimated
- Large SE $\Rightarrow$ wide range of plausible values
- More data $\Rightarrow$ smaller SE $\Rightarrow$ more certainty

SE answers: "if we repeated this study, how much would $\hat{w}$ vary?"

Is This Feature Significant? (Wald Test)

The Question: Does feature j actually matter, or is its effect just noise?

- H_0: $w_j = 0$ (feature has no effect)
- H_1: $w_j \neq 0$ (feature matters)

Wald Statistic (z-score):

$$z = \frac{\hat{w}_j}{\mathrm{SE}(\hat{w}_j)} \xrightarrow{H_0} \mathcal{N}(0,1) \tag{2}$$

Decision Rule:

- $|z| > 1.96$ (p-value < 0.05): coefficient is **significant**
- $|z| \leq 1.96$ (p-value ≥ 0.05): cannot conclude it matters

Always check p-values before interpreting coefficients in regulatory reports

Confidence Intervals for Odds Ratios

95% CI for Coefficient:

$$\hat{w}_j \pm 1.96 \times \text{SE}(\hat{w}_j) \tag{3}$$

95% CI for Odds Ratio (exponentiate both bounds):

$$\exp\big(\hat{w}_j \pm 1.96 \times \text{SE}(\hat{w}_j)\big) \tag{4}$$

Worked Example: Income coefficient $\hat{w} = 0.5$, SE $= 0.1$

- CI for w: [0.304, 0.696]
- CI for OR: $[e^{0.304}, e^{0.696}] = [1.36, 2.01]$
- Interpretation: income increases odds of approval by 36% to 101%

If CI for odds ratio contains 1.0, the effect is not statistically significant

Model Fit: Deviance and Pseudo-R^2

Deviance (badness-of-fit measure):

$$D = -2 \times \ell(\hat{\mathbf{w}}) \tag{5}$$

Two Key Benchmarks:

- **Null deviance** D_0: model with intercept only (baseline)
- **Residual deviance** D_1: model with all features

McFadden's Pseudo-R^2:

$$R^2_{\text{McFadden}} = 1 - \frac{D_1}{D_0} = 1 - \frac{\ell(\text{full})}{\ell(\text{null})} \tag{6}$$

Interpretation: values of 0.2–0.4 are considered good for logistic regression.

Deviance drop = model improvement; compare models via Likelihood Ratio Test

Likelihood Ratio Test (LRT)

Compares nested models: reduced (L_0) vs. full (L_1):

$$\Lambda = -2\left[\ell(\text{reduced}) - \ell(\text{full})\right] = D_{\text{reduced}} - D_{\text{full}} \;\sim\; \chi^2_{df} \tag{7}$$

where df = difference in number of parameters.

- More powerful than Wald test for testing multiple coefficients simultaneously
- Reject H_0 (reduced model adequate) if $\Lambda > \chi^2_{\alpha,\, df}$

Example: Adding 3 credit-bureau features to a PD model: $\Lambda = 15.7$, $df = 3$, $p = 0.0013 \Rightarrow$ significant improvement.

LRT is the standard for nested model comparison; use AIC/BIC for non-nested models

Model Selection: AIC and BIC

Information Criteria (for comparing models, including non-nested):

$$\text{AIC} = -2\ell + 2k, \qquad \text{BIC} = -2\ell + k \ln n \tag{8}$$

where k = number of parameters, n = sample size.

- **Lower is better** for both criteria
- AIC: asymptotically equivalent to leave-one-out CV
- BIC: penalizes complexity more; consistent (selects true model as $n \to \infty$)

Credit Scoring Context:

- BIC preferred for regulatory models (simpler = more interpretable)
- AIC for internal risk models where prediction accuracy matters

Combine LRT for nested models with AIC/BIC for non-nested comparisons

Threshold Selection

Default Threshold: 0.5

- Predict $\hat{y} = 1$ if $P(y = 1|\mathbf{x}) \geq 0.5$, equivalently $\mathbf{w}^T\mathbf{x} + b \geq 0$
- Minimizes misclassification rate when classes are balanced

Custom Thresholds (cost-based selection):

- Lower threshold $\Rightarrow$ higher recall (more sensitive)
- Higher threshold $\Rightarrow$ higher precision (more specific)
- Optimal threshold: minimize $C_{\text{FP}} \cdot \text{FP} + C_{\text{FN}} \cdot \text{FN}$

Example – Fraud Detection: Cost of missing fraud (FN) $\gg$ cost of false alarm (FP) $\Rightarrow$ use lower threshold (e.g., 0.3).

Optimal threshold depends on the cost matrix of your specific application

Non-Linear Boundaries via Feature Engineering

Polynomial Features:

- Original: $[x_1, x_2]$
- Expanded: $[x_1, x_2, x_1^2, x_2^2, x_1 x_2]$
- Creates curved decision boundaries in the original feature space

Trade-offs:

- More features $\Rightarrow$ more flexible boundaries
- Risk: overfitting to training data noise
- Solution: combine with regularization (L1/L2)

Model is linear in parameters but can capture non-linear patterns via engineered features

Multiclass Extension: Softmax

One-vs-Rest (OvR):

- Train K separate binary classifiers
- Predict class with highest probability

Multinomial (Softmax) Logistic Regression:

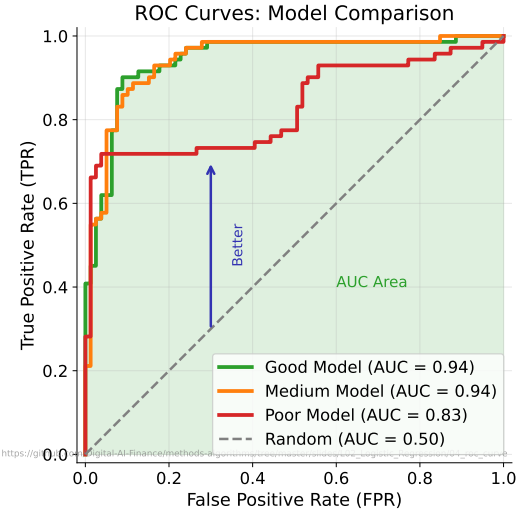
$$P(y = k|\mathbf{x}) = \frac{e^{\mathbf{w}_k^T \mathbf{x}}}{\sum_{j=1}^{K} e^{\mathbf{w}_j^T \mathbf{x}}}$$

- Single model; probabilities sum to 1 by construction
- Loss: categorical cross-entropy over K classes

---

scikit-learn: `multi_class='multinomial'` **for true softmax regression**

ROC Curve



ROC: plots TPR vs. FPR across all thresholds. Diagonal = random; upper-left = perfect.

ROC is threshold-independent – summarizes discrimination across all cutoffs

AUC Interpretation and Gini Coefficient

AUC Guidelines:

- 0.9–1.0: Excellent 0.8–0.9: Good 0.7–0.8: Fair
- AUC = probability that a random positive ranks higher than a random negative

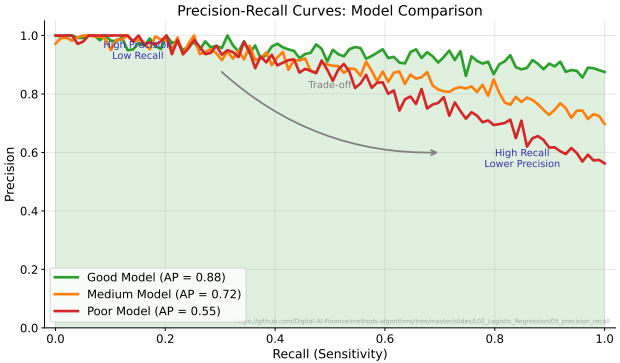
Gini Coefficient (finance standard):

$$\text{Gini} = 2 \times \text{AUC} - 1$$

- AUC $= 0.80 \Rightarrow$ Gini $= 0.60$
- AUC $= 0.75 \Rightarrow$ Gini $= 0.50$ (acceptable for credit scoring)
- Banks typically report Gini rather than AUC

Industry benchmarks: Gini > 0.40 **acceptable; Gini** > 0.60 **good**

Precision-Recall Curve



Key: Plots precision vs. recall at each threshold. More informative than ROC when the positive class is rare (e.g., fraud, default).

Use PR curves for imbalanced datasets where positive class is rare

ROC vs. Precision-Recall: When to Use Each

Use ROC When:

- Classes are roughly balanced
- You care equally about both classes
- Comparing models at a specific FPR

Use Precision-Recall When:

- Classes are imbalanced (fraud, rare default)
- Positive class is the focus
- High precision is required (e.g., alert systems)

ROC can be overly optimistic with imbalanced data – PR curves reveal the truth

Calibration and Brier Score

What is Calibration?

- Predicted 70% probability should mean ≈70% actually positive
- Well-calibrated: predicted probabilities match observed frequencies

Measuring Calibration:

- **Reliability diagram:** plot predicted vs. observed probability per bin
- **Brier score:** $BS = \frac{1}{n}\sum_{i=1}^{n}(\hat{p}_i - y_i)^2$ (lower is better)

Logistic Regression Advantage:

- Naturally well-calibrated (MLE directly optimizes log-likelihood)
- Unlike trees/forests that often require post-hoc calibration (Platt scaling)

Calibration is critical when probabilities drive financial decisions (e.g., PD for capital)

Hosmer-Lemeshow Goodness-of-Fit Test

Procedure:

1. Sort observations by $\hat{p}_i$; divide into G groups (typically $G = 10$)
2. For each group g: compare observed events O_g vs. expected $E_g = \sum_{i \in g} \hat{p}_i$

$$\hat{C} = \sum_{g=1}^{G} \frac{(O_g - E_g)^2}{E_g(1 - E_g/n_g)} \sim \chi^2_{G-2} \tag{9}$$

- Large $\hat{C}$ (small p-value) $\Rightarrow$ reject: model is poorly calibrated
- Sensitive to choice of G; combine with visual calibration plots
- Basel requires calibration testing for all PD models

Limitation: loses power with very large n. Consider le Cessie–van Houwelingen as alternative.

Why Regularize?

The Overfitting Problem:

- Many features, limited data $\Rightarrow$ model fits noise
- Perfect training accuracy but poor test performance

Solution: Penalize Large Coefficients

$$\mathcal{L}_{\text{reg}} = \underbrace{\mathcal{L}(\mathbf{w})}_{\text{data fit}} + \underbrace{\lambda \cdot R(\mathbf{w})}_{\text{complexity penalty}}$$

- $\lambda > 0$: regularization strength (hyperparameter)
- Larger $\lambda \Rightarrow$ simpler model (smaller coefficients)
- $\lambda = 0$: no regularization (standard MLE)

Regularization trades a small increase in bias for a large reduction in variance

L1 vs. L2 Regularization

L2 – Ridge:

$$\mathcal{L}_{\text{Ridge}} = \mathcal{L} + \lambda \sum_{j=1}^{p} w_j^2$$

- Shrinks all coefficients toward zero; keeps all features
- Good when most features are likely relevant

L1 – Lasso:

$$\mathcal{L}_{\text{Lasso}} = \mathcal{L} + \lambda \sum_{j=1}^{p} |w_j|$$

- Drives some coefficients to exactly zero
- Automatic feature selection – produces sparse models

L1 for feature selection; L2 when all features contribute

Elastic Net: Best of Both Worlds

Combined Penalty:

$$\mathcal{L}_{\text{EN}} = \mathcal{L} + \lambda \left[\alpha \sum |w_j| + (1 - \alpha) \sum w_j^2 \right]$$

where $\alpha \in [0, 1]$ controls the L1/L2 mix.

Advantages over Pure Lasso:

- Handles correlated features better (selects groups together)
- More stable feature selection across bootstrap samples
- Reduces to Ridge ($\alpha = 0$) or Lasso ($\alpha = 1$)

In scikit-learn: `LogisticRegression(penalty='elasticnet', solver='saga', l1_ratio=0.5)`

Elastic Net: `l1_ratio=1` **is pure L1;** `l1_ratio=0` **is pure L2**

Choosing λ via Cross-Validation

Procedure:

- Try grid of λ values: $[0.001, 0.01, 0.1, 1, 10, 100]$
- Use k-fold CV to estimate out-of-sample performance
- Select λ with best CV score (e.g., AUC or log-loss)

scikit-learn Convenience:

- `LogisticRegressionCV`: automatic λ search with built-in CV
- `Cs`: number of C values to try ($C = 1/\lambda$; larger C = less regularization)

LogisticRegressionCV handles cross-validation internally – recommended for production

Algorithm: Gradient Descent for Logistic Regression

Input: $\mathbf{X} \in \mathbb{R}^{n \times (p+1)}$, $\mathbf{y} \in \{0,1\}^n$, learning rate η, tolerance ϵ, max iterations T
Initialize $\mathbf{w} = \mathbf{0}$
for $t = 1$ to T **do**
  $\mathbf{p} = \sigma(\mathbf{X}\mathbf{w})$
  $\nabla = \frac{1}{n}\mathbf{X}^T(\mathbf{p} - \mathbf{y})$
  $\mathbf{w} = \mathbf{w} - \eta\nabla$
  if $\|\nabla\| < \epsilon$ **then**
    break
  end if
end for
return w

In practice, use L-BFGS or Newton-Raphson for faster convergence. Augment X with ones for bias.

Newton-Raphson / IRLS Optimization

Logistic regression is typically solved via **Iteratively Reweighted Least Squares** (IRLS):
Hessian (second derivative of log-likelihood):

$$\mathbf{H} = -\mathbf{X}^T\mathbf{W}\mathbf{X}, \quad \mathbf{W} = \text{diag}\big(p_i(1-p_i)\big) \tag{10}$$

Newton-Raphson update:

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} + (\mathbf{X}^T\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^T(\mathbf{y}-\mathbf{p}) \tag{11}$$

- Converges quadratically (much faster than gradient descent)
- Each iteration solves a weighted least squares problem (hence "IRLS")
- SEs come directly from $\mathbf{H}^{-1}$: no extra computation

This is what statsmodels and R's `glm()` compute. scikit-learn uses L-BFGS for speed.

Complete and Quasi-Complete Separation

The Problem: MLE fails when a hyperplane perfectly separates the classes.
Complete separation:

- A linear combination perfectly predicts all outcomes
- $\hat{w}_j \to \pm\infty$; algorithm fails to converge

Quasi-complete separation:

- Perfect separation except for a few tied points
- MLE exists but is unstable (inflated coefficients, huge SEs)

Solutions:

- **Firth's penalized likelihood**: adds Jeffreys prior to reduce bias
- **Regularization**: L2 penalty prevents coefficient explosion
- **Detection**: watch for non-convergence warnings or extreme coefficients

---

Common in small finance datasets (e.g., rare defaults with many predictors)

scikit-learn Implementation

Basic Usage:

- `from sklearn.linear_model import LogisticRegression`
- `model = LogisticRegression(max_iter=1000)`
- `model.fit(X_train, y_train)`
- `y_proba = model.predict_proba(X_test)[:, 1]`

Key Parameters:

- `C`: inverse regularization ($C = 1/\lambda$; default=1.0)
- `penalty`: `'l1'`, `'l2'`, `'elasticnet'`, `'none'`
- `solver`: `'lbfgs'` (default), `'liblinear'`, `'saga'`
- `class_weight`: set to `'balanced'` for imbalanced data

`predict_proba` **returns** $[P(y=0),\ P(y=1)]$ **– use** `[:, 1]` **for positive class probability**

Handling Class Imbalance

The Problem: 99% negatives, 1% positives $\Rightarrow$ predicting all negatives gives 99% accuracy.

Solutions:

- **Class weights**: `class_weight='balanced'` (sets $w_k \propto 1/n_k$)
- **Oversampling**: SMOTE or random oversampling of minority class
- **Undersampling**: random undersampling of majority class
- **Threshold tuning**: optimize for F1 or a business-specific cost metric

Weighted Cross-Entropy Loss:

$$\mathcal{L}_{\text{w}} = -\sum_{i=1}^{n} w_{y_i}[y_i \log p_i + (1 - y_i)\log(1 - p_i)]$$

Class weighting is usually the simplest and most effective first step

Credit Scoring in Practice

How Banks Use Logistic Regression:

- **PD (Probability of Default)**: probability a borrower will not repay
- Banks are *required* to estimate PD under Basel II/III regulations
- Logistic regression: industry standard (interpretable, auditable, stable)

Why Interpretability Matters:

- Regulators require explanation of every coefficient's contribution
- Must justify why income or debt ratio affects the approval decision
- Black-box models (neural networks) often rejected by supervisors

---

Basel IRB approach: banks must demonstrate PD model validity annually

From Coefficients to Scorecards

Key Industry Metrics:

- **Gini Coefficient**: $\text{Gini} = 2 \times \text{AUC} - 1$
  - $\text{AUC} = 0.75 \Rightarrow \text{Gini} = 0.50$ (acceptable)
  - $\text{AUC} = 0.85 \Rightarrow \text{Gini} = 0.70$ (good)
- **KS Statistic**: maximum separation between default and non-default CDFs

Scorecard Points:

- Convert log-odds to points: higher score = lower risk
- Typical: "each 20 points doubles the odds of being good"
- Points per feature $= -\left(\frac{\text{PDO}}{\ln 2}\right) \times w_j \times (\text{bin value})$

Industry practice: Gini > 0.40 acceptable; Gini > 0.60 good

Feature Engineering for Credit Scoring

Preprocessing for Logistic Regression:

- **Standardization**: mean $= 0$, std $= 1$ for all continuous features
- **Binning**: discretize continuous variables (e.g., age groups) for monotonic WoE
- **WoE encoding**: Weight of Evidence transforms for categorical features

Credit-Specific Features:

- Debt-to-income ratio (interaction: debt / income)
- Employment stability indicator (< 2 years flag)
- Bureau delinquency counts (non-linear effect $\Rightarrow$ bin)

Feature engineering often matters more than model selection in credit scoring

Hands-On Exercise

Interactive Notebook:

- Open: `notebooks/L02_logistic_regression.ipynb`
- Dataset: credit card fraud detection

Tasks:

1. Train logistic regression with L2 regularization
2. Evaluate with ROC, PR curves, and calibration plot
3. Tune classification threshold for business cost metric
4. Handle class imbalance with `class_weight='balanced'`
5. Interpret coefficients as odds ratios

Practice exercises reinforce mathematical concepts with real-world implementation

Key Takeaways

Mathematical Foundation:

- Sigmoid maps linear combination to probability; MLE via gradient descent
- Cross-entropy loss is convex $\Rightarrow$ guaranteed global optimum

Evaluation and Inference:

- Wald test, LRT, AIC/BIC for model selection and significance
- ROC/AUC for balanced data; PR curve for imbalanced; Gini for banking
- Calibration matters when probabilities drive decisions (PD models)

Practical:

- Regularization (Ridge/Lasso/Elastic Net) prevents overfitting
- Interpretability makes logistic regression the regulatory standard

---

Logistic regression: simple, fast, interpretable, and often competitive

Until Next Time...

"The sun has exploded – what's your posterior probability now?"

With logistic regression, you can quantify the answer.

Next Session: L03 – KNN & K-Means (from parametric to non-parametric methods)

XKCD #1132 callback – classification is about probabilities, not certainties

Appendix: Advanced Topics and Proofs

These slides are supplementary material for self-study

Newton-Raphson Convergence

Convergence Rate: For a twice-differentiable function with Lipschitz-continuous Hessian:

$$\|\boldsymbol{\beta}^{(t+1)} - \boldsymbol{\beta}^*\| \leq C \, \|\boldsymbol{\beta}^{(t)} - \boldsymbol{\beta}^*\|^2 \tag{12}$$

This is **quadratic convergence**: the number of correct digits doubles each iteration.

Requirements for Convergence:

- Hessian $\mathbf{H}$ must be negative definite at the optimum (guaranteed for logistic regression)
- Starting point must be sufficiently close to $\boldsymbol{\beta}^*$ (basin of attraction)
- No separation issues (Hessian becomes singular near separation)

Comparison: Gradient descent converges linearly: $\|\boldsymbol{\beta}^{(t+1)} - \boldsymbol{\beta}^*\| \leq \rho \, \|\boldsymbol{\beta}^{(t)} - \boldsymbol{\beta}^*\|$ with $\rho < 1$.

Newton converges in 5–10 iterations vs. hundreds for gradient descent

IRLS as Weighted Least Squares

Rewrite the Newton update as WLS:
Define the working response:

$$\mathbf{z}^{(t)} = \mathbf{X}\boldsymbol{\beta}^{(t)} + \mathbf{W}^{-1}(\mathbf{y} - \mathbf{p}^{(t)}) \tag{13}$$

Then the Newton update becomes:

$$\boldsymbol{\beta}^{(t+1)} = (\mathbf{X}^T\mathbf{W}^{(t)}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{W}^{(t)}\mathbf{z}^{(t)} \tag{14}$$

This is exactly the **weighted least squares** solution with weights $\mathbf{W}^{(t)} = \text{diag}\big(p_i^{(t)}(1 - p_i^{(t)})\big)$.

Insight: Each IRLS iteration solves a WLS problem with updated weights and working response. This connects GLM theory to standard linear regression machinery.

IRLS unifies logistic, Poisson, and other GLM estimation under one framework

Multinomial Logit: Softmax Gradient

Softmax for K classes:

$$P(y = k|\mathbf{x}) = \frac{e^{\mathbf{w}_k^T \mathbf{x}}}{\sum_{j=1}^{K} e^{\mathbf{w}_j^T \mathbf{x}}} \tag{15}$$

Categorical Cross-Entropy Loss:

$$\mathcal{L} = -\sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}[y_i = k] \log P(y_i = k|\mathbf{x}_i) \tag{16}$$

Gradient for class k:

$$\nabla_{\mathbf{w}_k} \mathcal{L} = \sum_{i=1}^{n} (P(y_i = k|\mathbf{x}_i) - \mathbb{1}[y_i = k])\, \mathbf{x}_i \tag{17}$$

Same elegant form as binary case: gradient = (predicted − actual) × features.

Binary logistic regression is the $K = 2$ special case of multinomial logit

Complete Separation: Firth's Penalized Likelihood

Firth's Modification (1993): add Jeffreys invariant prior as a penalty:

$$\ell^*(\boldsymbol{\beta}) = \ell(\boldsymbol{\beta}) + \frac{1}{2}\log\det(\mathbf{I}(\boldsymbol{\beta})) \tag{18}$$

where $\mathbf{I}(\boldsymbol{\beta}) = \mathbf{X}^T\mathbf{W}\mathbf{X}$ is the Fisher information matrix.

Modified Score Equation:

$$\mathbf{X}^T(\mathbf{y} - \mathbf{p} + \mathbf{h}) = \mathbf{0} \tag{19}$$

where $h_i = \frac{1}{2}\text{diag}(\mathbf{H})_i(1 - 2p_i)$ and $\mathbf{H}$ is the hat matrix.

Properties:

- Finite estimates even under complete separation
- Removes first-order bias: $E[\hat{\beta}^*] = \beta + O(n^{-2})$
- Available in R (`logistf`) and Python (`firthlogist`)

Firth's method is the gold standard for small-sample logistic regression

The Score (Rao) Test

Three Asymptotically Equivalent Tests:

- **Wald**: tests whether $\hat{\beta}$ is far from 0 (requires full model fit)
- **LRT**: compares log-likelihoods of nested models (requires both fits)
- **Score (Rao)**: tests whether gradient at H_0 is non-zero (requires *only null* fit)

Score Statistic:

$$S = \mathbf{U}(\boldsymbol{\beta}_0)^T \, \mathbf{I}(\boldsymbol{\beta}_0)^{-1} \, \mathbf{U}(\boldsymbol{\beta}_0) \;\sim\; \chi^2_q \tag{20}$$

where $\mathbf{U} = \nabla \ell$ is the score function and q is the number of restrictions.

Advantage: Only requires fitting the null (simpler) model – computationally cheaper for testing many candidate features.

Score test is the basis for forward stepwise variable selection in credit scoring

Prior on Coefficients:

$$\boldsymbol{\beta} \sim \mathcal{N}(\mathbf{0}, \tau^2 \mathbf{I}) \tag{21}$$

Posterior (via Bayes' theorem):

$$p(\boldsymbol{\beta}|\mathbf{X}, \mathbf{y}) \propto \underbrace{L(\boldsymbol{\beta})}_{\text{likelihood}} \times \underbrace{p(\boldsymbol{\beta})}_{\text{prior}} \tag{22}$$

MAP Estimate (maximum a posteriori):

$$\hat{\boldsymbol{\beta}}_{\text{MAP}} = \arg\max_{\boldsymbol{\beta}} \left[\ell(\boldsymbol{\beta}) - \frac{1}{2\tau^2} \|\boldsymbol{\beta}\|^2 \right] \tag{23}$$

Key Insight: MAP with Gaussian prior $\equiv$ L2-regularized MLE (Ridge) with $\lambda = 1/\tau^2$.

Full Bayesian inference (via MCMC or variational methods) provides complete posterior distributions and credible intervals for each coefficient.

Bayesian approach naturally handles separation and provides uncertainty quantification

References and Resources

Textbooks

- James, Witten, Hastie, Tibshirani (2021). *Introduction to Statistical Learning*. Ch. 4.
- Hastie, Tibshirani, Friedman (2009). *Elements of Statistical Learning*. Ch. 4.
- Hosmer, Lemeshow, Sturdivant (2013). *Applied Logistic Regression*. 3rd ed.
- Agresti (2013). *Categorical Data Analysis*. 3rd ed.

Online Resources

- scikit-learn: `https://scikit-learn.org/stable/modules/linear_model.html`
- statsmodels: `https://www.statsmodels.org/stable/generated/statsmodels.discrete.discrete_model.Logit.html`
- Stanford CS229: `https://cs229.stanford.edu/`

Primary textbook: ISLR Chapter 4; Applied Logistic Regression (Hosmer & Lemeshow) for depth