Logistic Regression

MSE vs. Cross Entropy, Likelihoods, and Gradient Descent

CS115B - Spring 2025

James Pustejovsky
Brandeis University

January 24, 2025

Today's Lecture

1. Linear Regression and MSE
2. Transition to Logistic Regression
3. Cross Entropy Loss
4. Gradient Descent (GD) for Logistic Regression
5. Worked-Out Example
6. Regularization: Ridge and Lasso

Motivation

Why start from linear regression?

- Linear regression is often the first introduction to parametric models.
- Understanding MSE and likelihood in linear regression sets the stage for logistic regression.
- Many foundational concepts (e.g., gradient descent) are introduced naturally in linear regression.

Linear Regression Setup

- We have a dataset $\{(x_i, y_i)\}_{i=1}^n$, where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$.
- Model assumption: $y_i \approx \beta_0 + \beta_1 x_{i1} + \cdots + \beta_d x_{id}$.
- In matrix form: $\mathbf{y} \approx X\boldsymbol{\beta}$, where X is $n \times (d+1)$ (including a column of ones for the intercept).

Likelihood in Linear Regression (Gaussian Assumption)

Assumption:

- $y_i = \mathbf{x_i}^\top \boldsymbol{\beta} + \epsilon_i$,
- $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ (i.i.d).

Likelihood:

$$p(\mathbf{y} \mid X, \boldsymbol{\beta}, \sigma^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(- \frac{\left(y_i - \mathbf{x_i}^\top \boldsymbol{\beta}\right)^2}{2\sigma^2} \right).$$

Taking the negative log-likelihood gives:

$$-\log p(\mathbf{y} \mid X, \boldsymbol{\beta}, \sigma^2) = \sum_{i=1}^{n} \frac{\left(y_i - \mathbf{x_i}^\top \boldsymbol{\beta}\right)^2}{2\sigma^2} + \text{const.}$$

Mean Squared Error (MSE)

Definition:

$$\mathrm{MSE}(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \mathbf{x_i}^\top \boldsymbol{\beta}\right)^2.$$

Connection to Maximum Likelihood:

- Minimizing MSE is equivalent to maximizing the Gaussian likelihood (under constant σ^2).
- Closed-form solution: $\boldsymbol{\beta} = (X^\top X)^{-1} X^\top \mathbf{y}$ (assuming $X^\top X$ invertible).

Summary: MSE in Linear Regression

- Linear regression is simple and has a direct closed-form solution.
- MSE is a natural choice when errors are (assumed) Gaussian.
- But what if the output y_i is not continuous (e.g., $\{0, 1\}$ in classification)?
- Next: Why MSE-based methods fail for classification.

Why Not Use Linear Regression for Classification?

Problem 1: Unbounded Outputs

- Linear regression can predict any real value, but classification requires predictions in $(0, 1)$ to interpret as probabilities.

Problem 2: Interpretability

- Probability constraints are violated if you use $y \approx \mathbf{x}^\top \boldsymbol{\beta}$ directly.
- MSE does not reflect the likelihood in a Bernoulli setting.

Problem 3: Non-convex (or poor) optimization landscape

- Minimizing MSE for classification can lead to slow convergence or local minima issues.

Logistic Regression: The Bernoulli Assumption

- Binary outcome $y_i \in \{0, 1\}$.
- Probability of $y_i = 1$ given $\mathbf{x_i}$:

$$p(y_i = 1 \mid \mathbf{x_i}, \boldsymbol{\beta}) = \sigma(\mathbf{x_i}^\top \boldsymbol{\beta}),$$

  where $\sigma(z)$ is the logistic function: $\sigma(z) = \frac{1}{1+e^{-z}}$.
- Then $y_i \sim \text{Bernoulli}(\sigma(\mathbf{x_i}^\top \boldsymbol{\beta}))$.

Logit (Log-Odds) Function

- Odds $= \frac{p}{1-p}$, so $\log(\text{Odds}) = \log\left(\frac{p}{1-p}\right)$.
- In logistic regression,

$$\log\left(\frac{p}{1-p}\right) = \mathbf{x_i}^\top \boldsymbol{\beta}.$$

- This ensures p remains in $(0, 1)$.
- A key difference: output is a probability, not a direct numeric value.

Likelihood in Logistic Regression

Bernoulli Likelihood:

$$p(y_i \mid \mathbf{x_i}, \boldsymbol{\beta}) = \sigma(\mathbf{x_i}^\top \boldsymbol{\beta})^{y_i} \left[1 - \sigma(\mathbf{x_i}^\top \boldsymbol{\beta})\right]^{(1-y_i)}.$$

Full Dataset Likelihood:

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} \sigma(\mathbf{x_i}^\top \boldsymbol{\beta})^{y_i} \left[1 - \sigma(\mathbf{x_i}^\top \boldsymbol{\beta})\right]^{(1-y_i)}.$$

Log-Likelihood:

$$\ell(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left[y_i \ln \sigma(\mathbf{x_i}^\top \boldsymbol{\beta}) + (1 - y_i) \ln\left(1 - \sigma(\mathbf{x_i}^\top \boldsymbol{\beta})\right) \right].$$

Why Not Use MSE for Logistic Regression?

- MSE does not align with the Bernoulli distribution assumption.
- Minimizing MSE can lead to suboptimal probability estimates (slower convergence).
- The log-likelihood derived from the Bernoulli assumption is more appropriate.

Hence, we use Cross Entropy (negative log-likelihood).

Cross Entropy: Motivation

- In classification, we measure how well our predicted distribution matches the true distribution.
- For a single Bernoulli trial, cross entropy is:

$$H(p, q) = -\Big[p \ln q + (1 - p) \ln(1 - q)\Big],$$

- For logistic regression, $p = y_i \in \{0, 1\}$ and $q = \sigma(\mathbf{x_i}^\top \beta)$.
- Minimizing cross entropy is equivalent to maximizing the Bernoulli log-likelihood.

Cross Entropy (Binary Classification)

$$\text{CrossEntropyLoss}(\boldsymbol{\beta}) = -\sum_{i=1}^{n}\Big[y_i \ln \sigma(\mathbf{x_i}^\top \boldsymbol{\beta}) + (1 - y_i)\ln\big(1 - \sigma(\mathbf{x_i}^\top \boldsymbol{\beta})\big)\Big].$$

Equivalently:

$$\text{CrossEntropyLoss}(\boldsymbol{\beta}) = -\ell(\boldsymbol{\beta}),$$

where $\ell(\boldsymbol{\beta})$ is the log-likelihood.

Properties of Cross Entropy

- Always non-negative.
- Zero if and only if predicted probability matches the true labels perfectly.
- Directly linked to the Bernoulli distribution assumption $\implies$ it's a proper scoring rule for classification.
- Typically yields better calibrated probabilities than MSE.

Why Gradient Descent?

- No closed-form solution for logistic regression (unlike linear regression).
- Must rely on numerical optimization.
- Gradient Descent is a first-order method that iteratively updates parameters in the direction of decreasing loss.

Gradient Descent: Core Algorithm

Generic update rule:

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \eta \nabla_{\boldsymbol{\beta}} \Big(\text{Loss}(\boldsymbol{\beta}^{(t)}) \Big),$$

where:

- η is the learning rate.
- $\nabla_{\boldsymbol{\beta}}$ is the gradient w.r.t. $\boldsymbol{\beta}$.
- Loss($\boldsymbol{\beta}$) here is Cross Entropy.

Gradient of Cross Entropy for Logistic Regression

$$\text{CrossEntropyLoss}(\boldsymbol{\beta}) = -\sum_{i=1}^{n}\Big[y_i \ln \sigma(\mathbf{x_i}^\top \boldsymbol{\beta}) + (1-y_i)\ln\big(1-\sigma(\mathbf{x_i}^\top \boldsymbol{\beta})\big)\Big].$$

$$\nabla_{\boldsymbol{\beta}}\text{CrossEntropyLoss}(\boldsymbol{\beta}) = \sum_{i=1}^{n}(\sigma(\mathbf{x_i}^\top \boldsymbol{\beta}) - y_i)\mathbf{x_i}.$$

Derivation Sketch:

- Use chain rule on $\sigma(z) = 1/(1+e^{-z})$.
- $\frac{d}{dz}\sigma(z) = \sigma(z)\big(1-\sigma(z)\big)$.
- Combine terms carefully from the log-likelihood expression.

Gradient Descent: Detailed Steps

Algorithm:

1. Initialize $\boldsymbol{\beta}^{(0)}$ (e.g., zeros or small random values).
2. For each iteration $t = 0, 1, 2, \ldots$:
   - Compute predictions: $\hat{y}_i^{(t)} = \sigma(\mathbf{x_i}^\top \boldsymbol{\beta}^{(t)})$.
   - Compute gradient: $\nabla_{\boldsymbol{\beta}} = \sum_{i=1}^{n} (\hat{y}_i^{(t)} - y_i)\mathbf{x_i}$.
   - Update parameters:
   $$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \eta \nabla_{\boldsymbol{\beta}}.$$
   - Check convergence criterion or iteration limit.

Challenges in Practice

- Choosing the learning rate η.
- Number of iterations vs. convergence speed.
- Computational cost when n is large (sum is over all training examples).

Solution: Stochastic Gradient Descent (SGD).

SGD: Basic Idea

- Instead of computing the gradient over the entire dataset, sample a small batch (or even a single point).
- Update parameters more frequently using noisy gradient estimates.
- Typically faster convergence in early stages; handles large datasets well.

Full Batch vs. Mini-Batch vs. Stochastic

- Batch Gradient Descent: Gradient computed using all n samples. - $\nabla_{\boldsymbol{\beta}} = \sum_{i=1}^{n}(\hat{y}_i - y_i)\mathbf{x_i}$.
- Stochastic Gradient Descent (Pure): Gradient computed using one sample at a time. - $\nabla_{\boldsymbol{\beta}} \approx (\hat{y}_{i(t)} - y_{i(t)})\mathbf{x_{i(t)}}$.
- Mini-Batch SGD: Gradient computed using a small batch of size B ($B \ll n$). - $\nabla_{\boldsymbol{\beta}} \approx \sum_{j=1}^{B}(\hat{y}_{i_j} - y_{i_j})\mathbf{x_{i_j}}$.

Why Use SGD in Logistic Regression?

- Large-scale data: Full gradient descent can be very slow.
- Online learning: Data might arrive in streams; SGD updates model incrementally.
- Regularization effect: Noise in gradient can help escape shallow local minima, has a mild regularizing effect.
- State-of-the-art in practice: Often used with advanced optimization variants (Adam, RMSProp, etc.).

SGD: Detailed Steps

Algorithm (Mini-Batch Form):

1. Initialize $\boldsymbol{\beta}^{(0)}$.
2. For each epoch (pass through the data):
   - Shuffle training data.
   - Partition training data into batches $\{B_1, \ldots, B_m\}$.
   - For each batch B_k:
     - Compute predictions for each (x_i, y_i) in B_k.
     - Compute gradient for that batch: $\nabla_{\boldsymbol{\beta}}^{(B_k)} = \sum_{i \in B_k} (\hat{y}_i - y_i)\mathbf{x_i}$.
     - Update:
       $$\boldsymbol{\beta}^{(\text{new})} = \boldsymbol{\beta}^{(\text{old})} - \eta \nabla_{\boldsymbol{\beta}}^{(B_k)}.$$
3. Evaluate convergence or stop criterion.

Dataset Description

Suppose we have a small synthetic dataset for a binary classification:

$$X = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} \\ 1 & x_{2,1} & x_{2,2} \\ \vdots & \vdots & \vdots \\ 1 & x_{n,1} & x_{n,2} \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad y_i \in \{0, 1\}.$$

- We have 2 features per example (plus the intercept). - Let's walk through one epoch of gradient descent step-by-step.

Step 1: Forward Pass (Compute Predictions)

For each i:

$$z_i^{(t)} = \mathbf{x_i}^\top \boldsymbol{\beta}^{(t)} = \beta_0^{(t)} + \beta_1^{(t)} x_{i,1} + \beta_2^{(t)} x_{i,2},$$

$$\hat{y}_i^{(t)} = \sigma(z_i^{(t)}) = \frac{1}{1 + e^{-z_i^{(t)}}}.$$

Example numeric placeholders:

- If $\beta_0^{(t)} = 0, \beta_1^{(t)} = 0.5, \beta_2^{(t)} = -0.3$,
- And $x_{i,1} = 2, x_{i,2} = 1.5$, then

$$z_i^{(t)} = 0 + 0.5 \cdot 2 + (-0.3) \cdot 1.5 = 1.0 - 0.45 = 0.55,$$

$$\hat{y}_i^{(t)} = \sigma(0.55) \approx 0.6341.$$

Step 2: Compute Loss & Gradient

$$\text{CrossEntropyLoss}(\boldsymbol{\beta}^{(t)}) = -\sum_{i=1}^{n}\left[y_i \ln\left(\hat{y}_i^{(t)}\right) + (1 - y_i)\ln\left(1 - \hat{y}_i^{(t)}\right)\right].$$

$$\nabla_{\boldsymbol{\beta}} = \sum_{i=1}^{n}(\hat{y}_i^{(t)} - y_i)\mathbf{x_i}.$$

Concrete Example: If $y_i = 1$ and $\hat{y}_i^{(t)} = 0.6341$, then the contribution to gradient is $(0.6341 - 1)\mathbf{x_i} = -0.3659\mathbf{x_i}$.

Step 3: Parameter Update

$$\beta_j^{(t+1)} = \beta_j^{(t)} - \eta \cdot \frac{\partial}{\partial \beta_j} \text{CrossEntropyLoss}(\boldsymbol{\beta}^{(t)}).$$

$$\beta_j^{(t+1)} = \beta_j^{(t)} - \eta \sum_{i=1}^{n} (\hat{y}_i^{(t)} - y_i) x_{i,j}.$$

Interpretation:

- If $\hat{y}_i^{(t)} > y_i$, we reduce the weight in proportion to $x_{i,j}$.
- If $\hat{y}_i^{(t)} < y_i$, we increase the weight in proportion to $x_{i,j}$.

Numerical Example Recap

- Start: $\beta_0^{(0)} = 0, \beta_1^{(0)} = 0.5, \beta_2^{(0)} = -0.3$.
- Learning rate $\eta = 0.1$.
- Suppose for one step we get $\nabla_{\boldsymbol{\beta}} = \begin{bmatrix} 0.2 \\ -1.1 \\ 0.5 \end{bmatrix}$ from the sum over n examples.
- Update:

$$\boldsymbol{\beta}^{(1)} = \boldsymbol{\beta}^{(0)} - 0.1 \begin{bmatrix} 0.2 \\ -1.1 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.5 \\ -0.3 \end{bmatrix} - \begin{bmatrix} 0.02 \\ -0.11 \\ 0.05 \end{bmatrix} = \begin{bmatrix} -0.02 \\ 0.61 \\ -0.35 \end{bmatrix}.$$

- Repeat until convergence.

Conclusion of Example

- Iteratively update the parameters.
- Model eventually converges to weights that (locally) maximize the likelihood (minimize cross-entropy).
- In practice, we often do mini-batch SGD for large datasets.

Summarizing

1. Linear Regression uses MSE, tied to a Gaussian likelihood assumption.
2. Logistic Regression is for classification with a Bernoulli assumption.
3. Cross Entropy (negative log-likelihood) is the natural loss for Bernoulli.
4. Gradient Descent is used since no closed-form solution exists for LR.
5. SGD scales better to large or streaming data, often used in modern ML pipelines.

Practical Considerations

- Choosing the right learning rate schedule or using advanced optimizers (Adam, Adagrad, RMSProp).
- Regularization (L2, L1) is often added to logistic regression to avoid overfitting.
- Feature engineering and scaling can significantly impact convergence.
- Evaluate model using metrics like accuracy, precision/recall, ROC AUC, etc.

Why Regularize?

Motivation: Overfitting

- A highly flexible model (many parameters) can perfectly fit the training data but perform poorly on new/unseen data.
- In linear/logistic regression, large coefficient values can make the model too sensitive to small changes (high variance).
- Regularization penalizes large weights to improve generalization.

Key Idea: Add a penalty term to the loss function that controls the magnitude of the parameters.

Ridge Regression (L2 Regularization)

Definition:

$$\text{Ridge Loss}(\boldsymbol{\beta}) = \underbrace{\text{MSE (or Cross Entropy)}}_{\text{Data fidelity}} + \underbrace{\lambda \sum_{j=1}^{d} \beta_j^2}_{\text{L2 penalty}},$$

- $\lambda \geq 0$ is the regularization strength (hyperparameter).
- The intercept (bias) is usually not regularized.

Interpretation:

- Penalizes the *square* of the weight magnitudes.
- Keeps the weight vector $\boldsymbol{\beta}$ “small” in Euclidean norm.
- Useful when many features contribute equally but you want to avoid very large coefficients.

Ridge Effects and Properties

Shrinkage Effect:

- Coefficients are “shrunk” towards zero, but usually no exact zeros.
- Ridge does not typically eliminate features—it just reduces their impact.

Geometric View:

- Adding $\lambda\|\beta\|_2^2$ (the squared L2 norm) to the loss constrains β to lie within a hypersphere in parameter space.
- Minimizing MSE subject to an L2 penalty is like finding an intersection of an “elliptical” MSE contour with a “circular” L2 boundary.

Hyperparameter Tuning:

- λ is usually chosen via cross-validation.
- Larger λ means stronger regularization (more coefficient shrinkage).

Lasso (L1 Regularization)

Definition:

$$\text{Lasso Loss}(\boldsymbol{\beta}) = \underbrace{\text{MSE (or Cross Entropy)}}_{\text{Data fidelity}} + \underbrace{\lambda \sum_{j=1}^{d} |\beta_j|}_{\text{L1 penalty}}.$$

- Similar to Ridge, but uses absolute values in the penalty.
- $\lambda \geq 0$ is again a hyperparameter controlling regularization strength.

Lasso Effects and Properties

Sparsity Effect:

- The L1 penalty can shrink some coefficients *exactly* to zero.
- Leads to sparse solutions (i.e., feature selection).

Geometric View:

- L1 penalty $\|\beta\|_1$ defines a diamond (or octahedron in higher dimensions).
- Minimizing the loss is like finding an intersection of the elliptical MSE contour with the corners of the diamond—those corners can force some coefficients to vanish.

Use Cases:

- When you expect many features are irrelevant or redundant and want the model itself to do feature selection.
- Often used when the number of features is large (high-dimensional data).

Comparing Ridge and Lasso

	Ridge (L2)	Lasso (L1)
Penalty form	$\lambda\|\boldsymbol{\beta}\|_2^2$	$\lambda\|\boldsymbol{\beta}\|_1$
Effect on coefficients	Shrinks them, rarely zero	Encourages zeros
Feature selection	No (dense solution)	Yes (sparse solution)
Typical usage	Prevents over-fitting	Feature sel., high-dim data

Elastic Net is a combination of both L1 and L2:

$$\text{Loss} = \text{MSE} + \lambda_1\|\boldsymbol{\beta}\|_1 + \lambda_2\|\boldsymbol{\beta}\|_2^2.$$

How They Are Used: Practical Steps

- Formulate the Regularized Loss:

$$\text{Loss} = \text{Data Fidelity Term} + \lambda\,(\text{Penalty}).$$

- Choose λ:
  - Often via cross-validation (e.g., grid-search).
  - Trade-off between model complexity and bias.
- Optimization:
  - For Ridge, you can still get a closed-form solution in linear regression.
  - For Lasso, no closed-form solution; use coordinate descent or other iterative methods (including gradient-based methods).
- Interpretation:
  - Ridge solutions yield non-zero weights for most features.
  - Lasso shrinks some weights exactly to zero, simplifying the model.

Summary of Regularization

- Regularization balances model fit with complexity control.
- Ridge (L2): Good for mitigating multicollinearity, keeps all features but with smaller coefficients.
- Lasso (L1): Encourages sparse solutions—automatically drops irrelevant features.
- Choosing λ is crucial—too large λ can underfit; too small λ may not sufficiently reduce overfitting.
- Applies broadly: linear regression, logistic regression, and beyond.

Further Reading

- Bishop, C. M., 2006. *Pattern Recognition and Machine Learning*.
- Murphy, K. P., 2012. *Machine Learning: A Probabilistic Perspective*.
- Hastie, T., Tibshirani, R., Friedman, J., 2009. *The Elements of Statistical Learning*.
- Online resources:
    - `https://www.deeplearningbook.org/`