Lesson 6.5: Logistic Regression

Mathematical Intuition · Module 6 — Capstone

Node 6.5

Overview

Complete decoding of logistic regression—the fundamental model for binary classification in machine learning.

The Core Equations

Model (Prediction)

$\hat{y} = \sigma(w^T x + b) = 1/(1 + e^{-(w^T x + b)})$

Loss Function (Binary Cross-Entropy)

$L(w,b) = -1/N \sum_i [y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$

Gradient Descent Update

$w \leftarrow w - \alpha \partial L/\partial w$
$b \leftarrow b - \alpha \partial L/\partial b$

Symbol-by-Symbol Breakdown

Input and Output

x (feature vector) - **Form**: $x = [x_1, x_2, \ldots, x\_d]^T$ - **Dimension**: d features - **Example**: [age, income, credit_score] - **Meaning**: Input data describing an instance

y (true label) - **Values**: 0 or 1 (binary classification) - **Example**: 0 = not spam, 1 = spam - **Meaning**: Ground truth class

$\hat{y}$ (predicted probability) - **Range**: [0, 1] - **Meaning**: P(y=1|x) (probability of class 1) - **Interpretation**: Confidence in positive class

Model Parameters

w (weight vector) - **Form**: $w = [w_1, w_2, \ldots, w\_d]^T$ - **Meaning**: Importance of each feature - **Learned**: Optimized during training - **Interpretation**: $w_i > 0 \rightarrow$ feature i increases probability of class 1

b (bias term) - **Scalar**: Single number - **Meaning**: Baseline log-odds (intercept) - **Learned**: Optimized during training - **Interpretation**: Shifts decision boundary

Linear Combination: `z = w^T x + b`

w^T x (dot product) - **Expansion**: $w^T x = w_1x_1 + w_2x_2 + \ldots + w\_dx\_d$ - **Meaning**: Weighted sum of features - **Units**: Depends on features and weights

z (logit or log-odds) - **Range**: $(-\infty, +\infty)$ - **Meaning**: Log of odds ratio - **Interpretation**: - $z > 0 \rightarrow$ more likely class 1 - $z < 0 \rightarrow$ more likely class 0 - $z = 0 \rightarrow$ equal probability

Sigmoid Function: σ(z) = 1/(1 + e^(-z))

σ (sigmoid activation) - **Domain**: z ∈ (-∞, +∞) - **Range**: σ(z) ∈ (0, 1) - **Properties**: - σ(0) = 0.5 - σ(z) → 1 as z → +∞ - σ(z) → 0 as z → -∞ - σ(-z) = 1 - σ(z)

Why sigmoid? - Squashes real line to [0,1] (valid probability) - Smooth and differentiable - Derivative: σ'(z) = σ(z)(1 - σ(z)) - Probabilistic interpretation (logistic distribution)

Alternative forms:

σ(z) = e^z/(1 + e^z)
σ(z) = 1/(1 + e^(-z))

Prediction: $\hat{y}$ = σ(w^T x + b)

Complete model:

$\hat{y}$ = P(y=1|x;w,b) = 1/(1 + e^(-(w^T x + b)))

Decision rule: - If $\hat{y} > 0.5$ → predict class 1 - If $\hat{y} < 0.5$ → predict class 0 - Threshold can be adjusted based on application

Interpretation: - Linear decision boundary in feature space - Boundary: w^T x + b = 0 - Probabilistic output (not just hard classification)

Loss Function

Binary Cross-Entropy: L = -1/N Σ_i[y_i log($\hat{y}_i$) + (1-y_i)log(1-$\hat{y}_i$)]

Breaking it down:

For single example:

ℓ($\hat{y}$,y) = -[y log($\hat{y}$) + (1-y)log(1-$\hat{y}$)]

If y = 1 (true class is 1):

ℓ = -log($\hat{y}$)

- Penalizes low predicted probability
- $\hat{y} \to 1$: loss → 0 (good)
- $\hat{y} \to 0$: loss → ∞ (bad)

If y = 0 (true class is 0):

ℓ = -log(1-$\hat{y}$)

- Penalizes high predicted probability
- $\hat{y} \to 0$: loss → 0 (good)
- $\hat{y} \to 1$: loss → ∞ (bad)

Average over dataset:

L = 1/N Σ_i ℓ($\hat{y}_i$, y_i)

- N: number of training examples
- Sum over all examples
- Average loss (mean)

Why Cross-Entropy?

Maximum likelihood: - Equivalent to maximizing likelihood of data - Assumes Bernoulli distribution for y - Convex loss function (single global minimum)

Gradient properties: - Smooth and differentiable - Gradients don't vanish (unlike squared error) - Penalizes confident wrong predictions heavily

Connection to KL divergence:

`L = D_KL(P_true || P_model) + H(P_true)`

- Minimizing cross-entropy = minimizing KL divergence
- H(P_true) is constant (entropy of true distribution)

Gradients

Gradient of Loss w.r.t. Weights

Chain rule:

$\partial L/\partial w = \partial L/\partial \hat{y} \cdot \partial \hat{y}/\partial z \cdot \partial z/\partial w$

Step 1: $\partial L/\partial \hat{y}$

$\partial L/\partial \hat{y} = -y/\hat{y} + (1-y)/(1-\hat{y})$

Step 2: $\partial \hat{y}/\partial z$ (sigmoid derivative)

$\partial \hat{y}/\partial z = \sigma(z)(1 - \sigma(z)) = \hat{y}(1-\hat{y})$

Step 3: $\partial z/\partial w$

$\partial z/\partial w = x$

Combining (beautiful cancellation):

$\partial L/\partial w = 1/N \sum_i (\hat{y}_i - y_i)x_i$

Interpretation: - Gradient = (prediction error) $\times$ (input) - Large error $\rightarrow$ large gradient - Correct prediction ($\hat{y} = y$) $\rightarrow$ zero gradient

Gradient of Loss w.r.t. Bias

Similarly:

$\partial L/\partial b = 1/N \sum_i (\hat{y}_i - y_i)$

Interpretation: - Average prediction error - Shifts decision boundary

Gradient Descent Update

Update Rules

Weights:

$w \leftarrow w - \alpha \, \partial L/\partial w = w - \alpha/N \sum_i (\hat{y}_i - y_i)x_i$

Bias:

$b \leftarrow b - \alpha \, \partial L/\partial b = b - \alpha/N \sum_i (\hat{y}_i - y_i)$

α (learning rate) - **Hyperparameter**: Controls step size - **Too large**: Oscillation, divergence - **Too small**: Slow convergence - **Typical values**: 0.001 to 0.1

Variants

Batch Gradient Descent: - Use all N examples per update - Exact gradient - Slow for large datasets

Stochastic Gradient Descent (SGD): - Use single example per update - Noisy gradient - Fast, online learning

Mini-batch Gradient Descent: - Use batch of B examples (e.g., B=32) - Balance between batch and SGD - Most common in practice

Complete Training Algorithm

Step 1: Initialize

$w \leftarrow$ random small values (e.g., N(0, 0.01))
$b \leftarrow 0$

Step 2: Forward Pass

For each example i:

$z_i = w^T x_i + b$
$\hat{y}_i = \sigma(z_i)$

Step 3: Compute Loss

$L = -1/N \sum_i [y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$

Step 4: Backward Pass (Compute Gradients)

$\partial L/\partial w = 1/N \sum_i (\hat{y}_i - y_i)x_i$
$\partial L/\partial b = 1/N \sum_i (\hat{y}_i - y_i)$

Step 5: Update Parameters

$w \leftarrow w - \alpha \partial L/\partial w$
$b \leftarrow b - \alpha \partial L/\partial b$

Step 6: Repeat

- Until convergence (loss stops decreasing)
- Or maximum iterations reached

Example: Spam Classification

Problem Setup

- **Features**: x = [word_count_“free”, word_count_“money”, email_length]
- **Label**: y = 1 (spam), y = 0 (not spam)
- **Goal**: Predict P(spam|email)

Training Data (simplified)

$x_1 = [5, 3, 100]$, $y_1 = 1$ (spam)
$x_2 = [0, 0, 200]$, $y_2 = 0$ (not spam)
$x_3 = [2, 1, 150]$, $y_3 = 1$ (spam)

After Training

```
w = [0.8, 0.6, -0.002]  (learned weights)
b = -1.5                 (learned bias)
```

Interpretation: - "free" increases spam probability ($w_1 = 0.8 > 0$) - "money" increases spam probability ($w_2 = 0.6 > 0$) - Longer emails slightly decrease spam probability (w_3 = -0.002 < 0) - Baseline bias: b = -1.5 (slightly biased toward not spam)

Prediction on New Email

```
x_new = [4, 2, 120]
z = w^T x_new + b = 0.8(4) + 0.6(2) - 0.002(120) - 1.5 = 2.46
ŷ = σ(2.46) = 1/(1 + e^(-2.46)) ≈ 0.92
```

Prediction: 92% probability of spam → classify as spam

Regularization

L2 Regularization (Ridge)

$\texttt{L\_reg = L + } \lambda \texttt{||w||}^2$

- Penalizes large weights
- Prevents overfitting
- λ: regularization strength

L1 Regularization (Lasso)

$\texttt{L\_reg = L + } \lambda \texttt{||w||}_1$

- Encourages sparsity (some $w_i = 0$)
- Feature selection
- λ: regularization strength

Elastic Net

$\texttt{L\_reg = L + } \lambda_1 \texttt{||w||}_1 \texttt{ + } \lambda_2 \texttt{||w||}^2$

- Combines L1 and L2
- Balance between sparsity and smoothness

Multiclass Extension: Softmax Regression

Model

$\texttt{P(y=k|x) = e^(w\_k^T x + b\_k) / } \Sigma_j \texttt{ e^(w\_j^T x + b\_j)}$

Loss (Cross-Entropy)

$\texttt{L = -1/N } \Sigma_i \ \Sigma_k \texttt{ y\_ik log(ŷ\_ik)}$

- y_ik: one-hot encoded (1 if example i is class k, else 0)
- ŷ_ik: predicted probability of class k

Practice Problems

1. **Sigmoid derivative**: Prove that σ'(z) = σ(z)(1 - σ(z)).
2. **Decision boundary**: For w = [1, 2], b = -3, find the equation of the decision boundary.

3. **Gradient**: Compute $\partial L/\partial w$ for a single example with x = [1, 2], y = 1, $\hat{y} = 0.7$.
4. **Convergence**: Why does cross-entropy loss lead to better convergence than squared error for classification?
5. **Regularization**: How does L2 regularization affect the gradient update?

Key Takeaways

- Logistic regression: Linear model + sigmoid activation
- Predicts probabilities: P(y=1|x) = σ(wˆT x + b)
- Cross-entropy loss: Penalizes wrong predictions
- Gradient descent: Iteratively updates w and b
- Convex optimization: Single global minimum
- Foundation for neural networks (single-layer network)
- Interpretable: Weights show feature importance

Historical Note

Logistic function introduced by Pierre François Verhulst (1838) - Originally for population growth models - **David Cox** (1958): Applied to regression (logistic regression) - Foundation of generalized linear models (GLMs) - Still widely used despite deep learning advances

Next Steps

Next, we'll decode the Fourier transform—the fundamental tool for signal processing and frequency analysis.

Mathematical Intuition · Module 6 — Capstone · Node 6.5
Concepts: cost-function, gradient-descent, likelihood, maximum-likelihood-estimation, inner-product

hbar.university — own.your.cognition