L02: Logistic Regression

Mathematical Foundations and Implementation

Methods and Algorithms – MSc Data Science

From Linear to Logistic

The Classification Problem

- Given features $\mathbf{x} \in \mathbb{R}^p$, predict $y \in \{0, 1\}$
- Linear regression: $\hat{y} = \mathbf{w}^T\mathbf{x} + b$ (unbounded)
- 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 function is also called the sigmoid function

The Sigmoid Function

01_sigmoid_function/chart.pdf

Odds and Log-Odds

Understanding the Model

- 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

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

Log-odds 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 = minimize negative log-likelihood (cross-entropy)

Binary Cross-Entropy Loss

03_log_loss/chart.pdf

Gradient Derivation

Computing the Gradient
For a single sample:

$$\frac{\partial \mathcal{L}}{\partial w_j} = (p - y)x_j$$

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 linear regression gradient!

The elegance of logistic regression: gradient has the same form as linear regression

Gradient Descent Update

Update Rule

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla_{\mathbf{w}} \mathcal{L}$$

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

Practical Considerations

- Feature scaling: standardize inputs for faster convergence
- Learning rate: start with $\eta = 0.01$, use line search or decay
- Convergence: monitor loss, check gradient norm ¡ tolerance

No closed-form solution like normal equation – must use iterative optimization

Linear Decision Boundary

02_decision_boundary/chart.pdf

Threshold Selection

Default Threshold: 0.5

- Predict 1 if $P(y = 1|\mathbf{x}) \geq 0.5$
- Equivalent to: $\mathbf{w}^T\mathbf{x} + b \geq 0$

Custom Thresholds

- Lower threshold: more sensitive (higher recall)
- Higher threshold: more specific (higher precision)
- Choose based on business costs of FP vs FN

Example: Fraud Detection

- Cost of missing fraud (FN) ¿¿ Cost of false alarm (FP)
- Use lower threshold, e.g., 0.3

Optimal threshold depends on the cost matrix of your application

Non-Linear Boundaries via Features

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

Trade-offs

- More features: more flexible boundaries
- Risk: overfitting to training data
- Solution: regularization

Logistic regression is linear in parameters, but can model non-linear boundaries

Multiclass Extension: Softmax

One-vs-Rest (OvR)

- Train K 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
- Loss: categorical cross-entropy

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

Confusion Matrix

06_confusion_matrix/chart.pdf

Classification Metrics

From the Confusion Matrix

- **Accuracy**: $\frac{TP+TN}{TP+TN+FP+FN}$ – overall correctness
- **Precision**: $\frac{TP}{TP+FP}$ – of predicted positives, how many correct?
- **Recall**: $\frac{TP}{TP+FN}$ – of actual positives, how many found?
- **F1 Score**: $\frac{2\cdot\text{Precision}\cdot\text{Recall}}{\text{Precision}+\text{Recall}}$

When Accuracy Fails

- Imbalanced data: 99% negative class
- Predicting all negatives gives 99% accuracy!

Accuracy is misleading for imbalanced datasets

ROC Curve

04_roc_curve/chart.pdf

Area Under ROC (AUC)

Interpretation

- AUC = 0.5: random guessing
- AUC = 1.0: perfect classifier
- AUC = probability(random positive ¿ random negative)

Guidelines

- 0.9–1.0: Excellent
- 0.8–0.9: Good
- 0.7–0.8: Fair
- 0.6–0.7: Poor
- 0.5–0.6: Fail

AUC is threshold-independent – summarizes performance across all thresholds

Precision-Recall Curve

05_precision_recall/chart.pdf

ROC vs Precision-Recall

When to Use ROC

- Balanced classes
- Care equally about both classes
- Comparing models at specific FPR

When to Use Precision-Recall

- Imbalanced classes (fraud, disease)
- Positive class is more important
- High precision required

ROC can be overly optimistic with imbalanced data

Calibration

What is Calibration?

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

Checking Calibration

- Reliability diagram (calibration plot)
- Brier score: $\frac{1}{n}\sum(p_i - y_i)^2$

Logistic Regression Advantage

- Naturally well-calibrated (MLE property)
- Unlike trees/random forests that may need calibration

Calibration is crucial when probabilities are used for decision-making

Regularization Motivation

The Overfitting Problem

- Many features, limited data
- Model fits noise, not signal
- Perfect training accuracy, poor test performance

Solution: Penalize Large Coefficients

$$\mathcal{L}_{\text{regularized}} = \mathcal{L} + \lambda \cdot \text{penalty}(\mathbf{w})$$

- λ: regularization strength (hyperparameter)
- Larger λ = simpler model

Regularization trades bias for variance

L1 vs L2 Regularization

L2 (Ridge)

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

- Shrinks coefficients toward zero
- Keeps all features, reduces magnitude

L1 (Lasso)

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

- Some coefficients exactly zero
- Automatic feature selection

L1 for sparse models, L2 when all features likely relevant

Elastic Net

Best of Both Worlds

$$\mathcal{L}_{\text{ElasticNet}} = \mathcal{L} + \lambda_1 \sum |w_j| + \lambda_2 \sum w_j^2$$

Advantages

- Handles correlated features better than Lasso alone
- Can select groups of correlated features
- More stable feature selection

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

Cross-Validation

- Try grid of λ values: [0.001, 0.01, 0.1, 1, 10, 100]
- Use k-fold CV to estimate test performance
- Select λ with best CV score

scikit-learn Convenience

- `LogisticRegressionCV`: automatic λ search
- `Cs`: inverse of λ (larger C = less regularization)

LogisticRegressionCV does cross-validation internally

Algorithm: Gradient Descent

```
1: Input: $\mathbf{X}$, $\mathbf{y}$, learning rate $\eta$, max iterations $T$
2: Initialize $\mathbf{w} = \mathbf{0}$
3: for $t = 1$ to $T$ do
4:     $\mathbf{p} = \sigma(\mathbf{X}\mathbf{w})$
5:     $\nabla = \frac{1}{n}\mathbf{X}^T(\mathbf{p} - \mathbf{y})$
6:     $\mathbf{w} = \mathbf{w} - \eta\nabla$
7:     if $\|\nabla\| < \epsilon$ then
8:         break
9:     end if
10: end for
11: return $\mathbf{w}$
```

In practice, use quasi-Newton methods (L-BFGS) for faster convergence

scikit-learn Implementation

Basic Usage

- `from sklearn.linear_model import LogisticRegression`
- `model = LogisticRegression()`
- `model.fit(X_train, y_train)`
- `y_pred = model.predict(X_test)`
- `y_proba = model.predict_proba(X_test)`

Key Parameters

- `C`: inverse regularization strength (default=1.0)
- `penalty`: 'l1', 'l2', 'elasticnet', 'none'
- `solver`: 'lbfgs', 'liblinear', 'saga'
- `class_weight`: 'balanced' for imbalanced data

predict_proba returns [P(y=0), P(y=1)] – use [:, 1] for positive class

Handling Class Imbalance

The Problem

- 99% negatives, 1% positives
- Model predicts all negatives: 99% accuracy!

Solutions

- **Class weights**: `class_weight='balanced'`
- **Oversampling**: SMOTE, random oversampling
- **Undersampling**: random undersampling
- **Threshold tuning**: optimize for F1 or business metric

Weighted Loss

$$\mathcal{L}_{\text{weighted}} = -\sum_i w_{y_i} [y_i \log p_i + (1 - y_i) \log(1 - p_i)]$$

class_weight='balanced' sets $w_k \propto 1/n_k$

Feature Engineering Tips

For Logistic Regression

- **Standardization**: mean=0, std=1 for all features
- **Missing values**: impute or create indicator variable
- **Categorical**: one-hot encoding (drop one level)
- **Interactions**: $x_1 \times x_2$ if domain suggests
- **Non-linearity**: binning or polynomial features

Credit Scoring Example

- Age: may have non-linear effect (bin into groups)
- Debt-to-income ratio: interaction of two features
- Employment length: indicator for ¡ 2 years

Feature engineering often matters more than model selection

Model Interpretation

Coefficient Analysis

- Sign: direction of effect
- Magnitude: strength (after standardization)
- Odds ratio e^{w_j}: multiplicative effect

Example Interpretation

- $w_{\text{income}} = 0.5$: each \$1000 income increase multiplies odds of approval by $e^{0.5} = 1.65$
- $w_{\text{debt\_ratio}} = -1.2$: each 0.1 increase in debt ratio multiplies odds by $e^{-0.12} = 0.89$ (11% decrease)

This interpretability makes logistic regression preferred in regulated industries

Solver Selection

Available Solvers in scikit-learn

- `lbfgs`: default, works for L2 and no penalty
- `liblinear`: fast for small data, supports L1
- `saga`: supports all penalties, works for large data
- `newton-cg`: similar to lbfgs
- `sag`: stochastic, for very large data

Guidelines

- L1 penalty: use liblinear or saga
- Large data: saga or sag
- Default (L2): lbfgs

solver='saga' is the most versatile but may be slower for small datasets

Convergence Issues

Warning: "Convergence Warning"

- Model did not converge in max_iter iterations
- May mean poor solution

Solutions

- Increase `max_iter` (default=100)
- Standardize features
- Increase regularization (smaller C)
- Use different solver

Always check for convergence warnings in production code

Logistic vs Other Classifiers

Strengths of Logistic Regression

- Interpretable coefficients
- Well-calibrated probabilities
- Fast training and prediction
- Works well with few samples

Limitations

- Linear decision boundary
- May underfit complex patterns
- Sensitive to outliers (compared to trees)

When to Choose Alternatives

- Complex patterns: Random Forests, Gradient Boosting
- High-dimensional: SVM with RBF kernel
- Interpretability not required: Neural Networks

Start with logistic regression as baseline, then try more complex models

Decision Framework

07_decision_flowchart/chart.pdf

Key Takeaways

Mathematical Foundation

- Sigmoid function maps linear combination to probability
- Maximum likelihood estimation via gradient descent
- Cross-entropy loss is convex, guaranteed global optimum

Evaluation

- Use confusion matrix, precision, recall, F1
- ROC/AUC for balanced data, PR curve for imbalanced
- Calibration matters when using probabilities

Practice

- Regularization prevents overfitting
- Class weights handle imbalance
- Coefficients are directly interpretable

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

References

Textbooks

- James et al. (2021). *ISLR*, Chapter 4: Classification
- Hastie et al. (2009). *ESL*, Chapter 4: Linear Methods

Documentation

- scikit-learn: LogisticRegression user guide
- statsmodels: Logit for statistical inference

Next Lecture

- L03: KNN and K-Means
- From parametric to non-parametric methods