Lecture 15: Multiclass Classification, ROC/AUC, and Introduction to Bayesian Inference

CS109A: Introduction to Data Science

Harvard University

- **Course:** CS109A: Introduction to Data Science
- **Lecture:** Lecture 15
- **Instructors:** Pavlos Protopapas, Kevin Rader, Chris Tanner
- **Topics:** Multinomial Logistic Regression, One-vs-Rest, Softmax, Confusion Matrix, ROC Curves, AUC, Bayesian Inference, Beta Distribution, Beta-Binomial Model

Key Summary

This lecture extends classification to the multiclass setting and introduces tools for evaluating classifier performance. We then transition to Bayesian statistics, laying groundwork for Bayesian approaches to regression.

Part 1 - Multiclass Classification:

- Multinomial Logistic Regression: Using a reference class
- One-vs-Rest (OvR): Building K separate binary classifiers
- Softmax function: Converting scores to proper probabilities
- Making predictions when K > 2

Part 2 - Classification Evaluation:

- Confusion matrices: TP, FP, TN, FN
- Threshold selection and trade-offs
- ROC curves and AUC for model comparison

Part 3 - Bayesian Introduction:

- Bayes' Rule for parameter estimation
- The Beta distribution as a conjugate prior
- The Beta-Binomial model
- Preview of hierarchical models

Contents

1 Review: From Binary to Multiclass

In the previous lectures, we focused on **binary classification**: predicting whether $Y = 0$ or $Y = 1$. Logistic regression models the probability of success:

$$P(Y = 1|X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

But what if Y can take on **more than two values**?

Example:

Real-World Multiclass Problems

- **Student major prediction**: CS, Statistics, or Other
- **NFL play type**: Pass, Run, or Special Teams
- **Email classification**: Primary, Social, Promotions, or Spam
- **Medical diagnosis**: Healthy, Condition A, Condition B, Condition C

We need to extend logistic regression to handle $K > 2$ classes. There are two main approaches:

1. **Multinomial Logistic Regression**: Compare each class to a reference class
2. **One-vs-Rest (OvR)**: Build K separate binary classifiers

2 Multinomial Logistic Regression

2.1 The Setup

Suppose we have K classes labeled $0, 1, 2, \dots, K-1$. Multinomial logistic regression:

1. Designates one class as the **reference group** (typically class $K-1$ or class 0)
2. Fits $K-1$ separate log-odds models comparing each other class to the reference

Example:

NFL Play Prediction We want to predict play type with $K = 3$ classes:

- Class 0: Special Teams (punt, field goal, etc.)
- Class 1: Pass play
- Class 2: Run play

Using **Class 0 as reference**, we fit two models:

Model 1 (Pass vs Special Teams):

$$\log\left(\frac{P(Y=1)}{P(Y=0)}\right) = \beta_0^{(1)} + \beta_1^{(1)} \cdot \text{Down} + \beta_2^{(1)} \cdot \text{Distance}$$

Model 2 (Run vs Special Teams):

$$\log\left(\frac{P(Y=2)}{P(Y=0)}\right) = \beta_0^{(2)} + \beta_1^{(2)} \cdot \text{Down} + \beta_2^{(2)} \cdot \text{Distance}$$

2.2 From Two Models to Three Probabilities

We have two equations but need three probabilities: $P(Y=0), P(Y=1), P(Y=2)$.

The key insight: **Probabilities must sum to 1**!

$$P(Y=0) + P(Y=1) + P(Y=2) = 1$$

With this third equation, we can solve for all three probabilities:

1. From Model 1: $P(Y=1) = P(Y=0) \cdot e^{\beta^{(1)}X}$
2. From Model 2: $P(Y=2) = P(Y=0) \cdot e^{\beta^{(2)}X}$
3. Sum constraint: $P(Y=0) + P(Y=0) \cdot e^{\beta^{(1)}X} + P(Y=0) \cdot e^{\beta^{(2)}X} = 1$

Solving for $P(Y=0)$:

$$P(Y=0) = \frac{1}{1 + e^{\beta^{(1)}X} + e^{\beta^{(2)}X}}$$

And then:

$$P(Y=k) = \frac{e^{\beta^{(k)}X}}{1 + e^{\beta^{(1)}X} + e^{\beta^{(2)}X}} \quad \text{for } k \in \{1, 2\}$$

2.3 Interpreting the Output

Example:

Reading sklearn's Multinomial Output sklearn reports coefficients for **all K classes** (not just K-1):

```python
model = LogisticRegression(multi_class='multinomial')
model.fit(X, y)
print(model.coef_)  # Shape: (3, 2) for 3 classes, 2 features
```

Output might look like:

```
[[-6.22,  1.67,  0.10],   # Class 0 coefficients
 [ 1.86, -0.04, -0.02],   # Class 1 coefficients
 [-1.64, -0.63, -0.08]]   # Class 2 coefficients
```

Why 3 sets of coefficients when theory says K-1?
sklearn internally **renormalizes** the coefficients so that each can be interpreted as "Class k vs Not Class k" rather than "Class k vs Reference." This makes predictions easier to compute but changes the interpretation slightly.
For the first class (Special Teams):

$$\log\left(\frac{P(Y=0)}{P(Y\neq 0)}\right) = -6.22 + 1.67 \cdot \text{Down} + 0.10 \cdot \text{Distance}$$

Interpretation: As "Down" increases (approaching 4th down), the probability of a special teams play increases. As "Distance" increases, special teams also becomes more likely (punting on 4th and long).

3 One-vs-Rest (OvR) Classification

3.1 The Approach

One-vs-Rest takes a different strategy: instead of comparing to a reference group, we build **K completely separate binary classifiers**, one for each class.

- **Classifier 1**: Class 0 vs (Classes 1, 2, ..., K-1)
- **Classifier 2**: Class 1 vs (Classes 0, 2, ..., K-1)
- ⋮
- **Classifier K**: Class K-1 vs (Classes 0, 1, ..., K-2)

Example:

OvR for NFL Plays **Classifier 1 (Special Teams vs Everything Else):**

- Positive class: Special Teams (Class 0)
- Negative class: Pass + Run (Classes 1, 2 combined)

Classifier 2 (Pass vs Everything Else):

- Positive class: Pass (Class 1)
- Negative class: Special Teams + Run (Classes 0, 2 combined)

Classifier 3 (Run vs Everything Else):

- Positive class: Run (Class 2)
- Negative class: Special Teams + Pass (Classes 0, 1 combined)

3.2 The Problem: Probabilities Don't Sum to 1

Each classifier outputs its own probability:

- $p_0 = P(\text{Special Teams vs Others})$
- $p_1 = P(\text{Pass vs Others})$
- $p_2 = P(\text{Run vs Others})$

But these three probabilities typically **do not sum to 1**! They come from independent models, each trained on slightly different data configurations.

3.3 Solution: The Softmax Function

To convert these scores into proper probabilities, we use the **Softmax function**:

Definition:

Softmax Function Given scores $(s_1, s_2, \ldots, s_K)$ for K classes, the softmax function converts them to probabilities:

$$P(Y = k) = \frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}}$$

Properties:

- All outputs are positive (due to exponential)
- All outputs are between 0 and 1
- All outputs **sum to exactly 1**
- Larger scores get larger probabilities (monotonic)

Example:

Softmax in Action Suppose our three classifiers output log-odds (scores):

- $s_0 = -2$ (Special Teams)
- $s_1 = 1.5$ (Pass)
- $s_2 = 0.8$ (Run)

Apply softmax:

$$\sum e^{s_j} = e^{-2} + e^{1.5} + e^{0.8} = 0.135 + 4.48 + 2.23 = 6.84$$

$$P(Y = 0) = \frac{0.135}{6.84} = 0.02$$
$$P(Y = 1) = \frac{4.48}{6.84} = 0.66$$
$$P(Y = 2) = \frac{2.23}{6.84} = 0.33$$

Now the probabilities sum to 1, and we'd predict **Pass** (Class 1).

3.4 Multinomial vs OvR: Which to Use?

Aspect	**Multinomial**	**OvR**
Number of models	$K - 1$	K
Training	Joint optimization	Independent classifiers
Data usage	All data, structured	May discard some structure
Decision boundaries	One per pair (vs reference)	One per class (vs all)
Performance	Often similar	Often similar

Key Information

How to Choose?

In practice, both methods often give very similar results. The best approach:

1. Try both methods
2. Compare using **cross-validation**
3. Choose based on test set performance (not training performance!)

4 Making Predictions with K > 2 Classes

4.1 From Probabilities to Classifications

In binary classification, we used the threshold $P(Y = 1) > 0.5$ to predict class 1.

With $K > 2$ classes, no single class is guaranteed to have probability > 0.5. Instead, we use the **plurality rule**:

$$\hat{Y} = \arg\max_k P(Y = k|X)$$

Simply predict the class with the highest probability, even if that probability is below 0.5.

Example:

Plurality Prediction Suppose our model predicts:

- $P(\text{Special Teams}) = 0.05$
- $P(\text{Pass}) = 0.55$
- $P(\text{Run}) = 0.40$

Prediction: Pass (highest probability)

Another example:

- $P(\text{Special Teams}) = 0.10$
- $P(\text{Pass}) = 0.45$
- $P(\text{Run}) = 0.45$

Prediction: Either Pass or Run (tie—implementation dependent)

4.2 Class Imbalance Problems

Warning

When Classification Always Predicts the Same Class

If one class dominates the data (e.g., 66% of NFL plays are passes), the model might predict "Pass" for almost every observation—and still achieve 66% accuracy!

The cocaine example from lecture: If you're predicting whether someone is currently high on cocaine, you'd predict "No" for everyone and be right 99.9%+ of the time.

Key insight: The model might still capture meaningful relationships through the **probabilities**, even if the pure **classifications** are all the same. Always examine predicted probabilities, not just classifications.

4.3 Loss Function for Multiclass

Binary classification uses **Binary Cross-Entropy**:

$$\text{Loss} = -\sum_{i=1}^{n} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

Multiclass classification generalizes this to **Cross-Entropy** (or Multinomial Logistic Loss):

$$\text{Loss} = -\sum_{i=1}^{n}\sum_{k=1}^{K} \mathbf{1}_{[y_i=k]} \log(p_{ik})$$

where p_{ik} is the predicted probability that observation i belongs to class k.

Regularization (L1/Lasso or L2/Ridge) can still be applied to this loss function to prevent overfitting.

5 Review: The Confusion Matrix

The **confusion matrix** is the foundation of classification evaluation. For binary classification:

		Predicted	
		Negative (0)	Positive (1)
Actual	Negative (0)	TN	FP
	Positive (1)	FN	TP

Definition:

Confusion Matrix Elements

- **True Positive (TP)**: Actually positive, predicted positive. Correct!
- **True Negative (TN)**: Actually negative, predicted negative. Correct!
- **False Positive (FP)**: Actually negative, predicted positive. Type I Error.
- **False Negative (FN)**: Actually positive, predicted negative. Type II Error.

Example:

Heart Disease Example From the lecture, predicting heart disease using age, sex, and their interaction:

	Predicted: No	Predicted: Yes
Actual: No	110 (TN)	54 (FP)
Actual: Yes	53 (FN)	86 (TP)

Is this a useful model?

- Among actual negatives ($110 + 54 = 164$): $110/164 = 67\%$ correctly identified
- Among actual positives ($53 + 86 = 139$): $86/139 = 62\%$ correctly identified
- Better than random chance (which would be 50/50)
- Not perfect, but has discriminatory power

6 The Threshold Trade-off

Logistic regression outputs **probabilities**, not classifications. We convert to classifications using a **threshold**:

$$\hat{Y} = \begin{cases} 1 & \text{if } \hat{p} \geq \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$

The default threshold is 0.5, but this can be changed!

6.1 Effect of Changing the Threshold

Lowering the threshold (e.g., 0.5 $\rightarrow$ 0.4):

- Model predicts "positive" more easily
- **TP increases** (good), **FN decreases** (good)
- **FP increases** (bad), **TN decreases** (bad)
- **Use case**: Medical screening—we don't want to miss sick patients (minimize FN)

Raising the threshold (e.g., 0.5 $\rightarrow$ 0.6):

- Model predicts "positive" more conservatively
- **FP decreases** (good), **TN increases** (good)
- **TP decreases** (bad), **FN increases** (bad)
- **Use case**: Spam filtering—we don't want to lose important emails (minimize FP)

Important:

The Fundamental Trade-off **You cannot simultaneously minimize both FP and FN.** Reducing one type of error inevitably increases the other. The optimal threshold depends on the **relative costs** of each type of error in your specific application.

7 ROC Curves

The **ROC Curve** (Receiver Operating Characteristic) visualizes classifier performance across **all possible thresholds**.

7.1 Key Metrics

Definition:

TPR and FPR **True Positive Rate (TPR)** = Sensitivity = Recall:

$$\text{TPR} = \frac{TP}{TP + FN} = P(\hat{Y} = 1 | Y = 1)$$

"Of all actual positives, what fraction did we catch?"
False Positive Rate (FPR) = 1 - Specificity:

$$\text{FPR} = \frac{FP}{FP + TN} = P(\hat{Y} = 1 | Y = 0)$$

"Of all actual negatives, what fraction did we falsely classify as positive?"

7.2 Constructing the ROC Curve

1. For each possible threshold (from 0 to 1):
   - Classify all observations using that threshold
   - Calculate the resulting TPR and FPR
   - Plot the point (FPR, TPR)
2. Connect all points to form the curve

7.3 Interpreting the ROC Curve

Key Reference Points:

- **(0, 0)**: Threshold = 1 (predict everyone negative)
- **(1, 1)**: Threshold = 0 (predict everyone positive)
- **(0, 1)**: Perfect classifier (100% TPR, 0% FPR)

Key Reference Lines:

- **Diagonal (y = x)**: Random classifier. A coin flip with probability p gives point (p, p).
- **Upper-left corner**: Ideal. We want the curve to hug this corner.

Example:

Reading ROC Curves If you see three curves:
- **Blue curve**: Hugs upper-left corner tightly
- **Green curve**: Moderately above the diagonal

- **Red dashed line**: The diagonal (random baseline)

The blue model is best—for any given FPR, it achieves higher TPR than the others.

8 AUC: Area Under the Curve

Comparing ROC curves visually can be difficult, especially when curves cross. The **AUC** (Area Under the Curve) summarizes performance in a single number.

8.1 Computing AUC

AUC is literally the area under the ROC curve:

$$\text{AUC} = \int_0^1 \text{ROC}(x)\, dx$$

In practice, we approximate this using the trapezoidal rule over the discrete threshold points.

8.2 Interpreting AUC

- **AUC = 1.0**: Perfect classifier
- **AUC = 0.5**: Random classifier (no better than coin flip)
- **AUC < 0.5**: Worse than random (predictions are inverted)

Key Information

Probabilistic Interpretation of AUC
AUC equals the probability that a randomly chosen positive example is ranked higher (assigned higher predicted probability) than a randomly chosen negative example.
AUC = 0.8 means: If you pick one positive and one negative case at random, there's an 80% chance the model assigns higher probability to the positive case.

```python
from sklearn.metrics import roc_curve, roc_auc_score

# Get predicted probabilities
y_proba = model.predict_proba(X_test)[:, 1]

# Calculate ROC curve points
fpr, tpr, thresholds = roc_curve(y_test, y_proba)

# Calculate AUC
auc_score = roc_auc_score(y_test, y_proba)
print(f"AUC: {auc_score:.3f}")
```

Listing 1: Computing ROC and AUC in sklearn

9 Introduction to Bayesian Inference

So far, we've used the **frequentist** approach:

- Parameters (β) are fixed but unknown constants
- We estimate them using data (MLE, OLS)
- Uncertainty is expressed through confidence intervals

The **Bayesian** approach takes a fundamentally different view:

Definition:

Bayesian Philosophy In Bayesian statistics, parameters are **random variables** with probability distributions.
We start with a **prior belief** about the parameter, observe data, and update our belief to obtain a **posterior distribution**.

9.1 Bayes' Rule for Parameter Estimation

$$\underbrace{f(\theta|X)}_{\text{Posterior}} = \frac{\overbrace{f(X|\theta)}^{\text{Likelihood}} \cdot \overbrace{f(\theta)}^{\text{Prior}}}{\underbrace{f(X)}_{\text{Normalizing constant}}}$$

In practice, we often ignore the normalizing constant and write:

$$\text{Posterior} \propto \text{Likelihood} \times \text{Prior}$$

The components:

- **Prior** $f(\theta)$: Our belief about θ *before* seeing the data
- **Likelihood** $f(X|\theta)$: Probability of observing data X if θ is the true value
- **Posterior** $f(\theta|X)$: Our updated belief about θ *after* seeing the data

Key Summary

Bayesian Inference in One Sentence:
Prior belief $\times$ Evidence from data $\rightarrow$ Updated belief

10 The Beta Distribution

Before we can do Bayesian inference for classification, we need a distribution for modeling probabilities. Enter the **Beta distribution**.

10.1 Why Beta?

We need a distribution that:

1. Is defined on $[0, 1]$ (since probabilities are between 0 and 1)
2. Is flexible enough to represent different prior beliefs
3. Works nicely with Bernoulli/Binomial likelihoods

The Beta distribution satisfies all three!

10.2 The Beta Distribution

Definition:

Beta Distribution A random variable $X \sim \text{Beta}(\alpha, \beta)$ has PDF:

$$f(x|\alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}, \quad x \in [0, 1]$$

where $\Gamma(\cdot)$ is the gamma function (generalization of factorial: $\Gamma(n) = (n-1)!$ for integers).

Key properties:

- Mean: $E[X] = \frac{\alpha}{\alpha+\beta}$
- Mode: $\frac{\alpha-1}{\alpha+\beta-2}$ (for $\alpha, \beta > 1$)
- Variance: $\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$

10.3 Shape of Beta Distributions

- **Beta**$(1, 1)$: Uniform distribution on $[0, 1]$. “I have no prior information.”
- **Beta**$(10, 10)$: Symmetric, peaked at 0.5. “I believe $p \approx 0.5$.”
- **Beta**$(2, 5)$: Skewed left, mean ≈ 0.29. “I believe p is small.”
- **Beta**$(5, 2)$: Skewed right, mean ≈ 0.71. “I believe p is large.”
- **Beta**$(0.5, 0.5)$: U-shaped. “I believe p is either very small or very large.”

Key Information

Intuitive Interpretation

Think of $\alpha - 1$ as “prior successes” and $\beta - 1$ as “prior failures” you’ve imagined before seeing real data.

$\text{Beta}(1, 1) = 0$ prior successes, 0 prior failures (no information)

$\text{Beta}(10, 10) = 9$ prior successes, 9 prior failures (believe coin is fair)

11 The Beta-Binomial Model

Now we put it all together: using the Beta distribution as a prior for a Binomial likelihood.

11.1 The Setup

Goal: Estimate the probability p of success (e.g., probability a coin lands heads).

Data: n independent trials with k successes and $n-k$ failures.

Model:

- **Likelihood**: $X|p \sim \text{Binomial}(n, p)$
- **Prior**: $p \sim \text{Beta}(\alpha_0, \beta_0)$

11.2 Computing the Posterior

Using Bayes' rule:

$$f(p|X) \propto f(X|p) \cdot f(p)$$

Likelihood (ignoring constants not involving p):

$$f(X|p) \propto p^k (1-p)^{n-k}$$

Prior:

$$f(p) \propto p^{\alpha_0 - 1} (1-p)^{\beta_0 - 1}$$

Posterior:

$$\begin{aligned} f(p|X) &\propto p^k (1-p)^{n-k} \cdot p^{\alpha_0 - 1} (1-p)^{\beta_0 - 1} \\ &= p^{(\alpha_0 + k) - 1} (1-p)^{(\beta_0 + n - k) - 1} \end{aligned}$$

This is a **Beta distribution**!

Important:

Beta-Binomial Update Rule

$$\begin{aligned} \text{Prior}: &\quad p \sim \text{Beta}(\alpha_0, \beta_0) \\ \text{Data}: &\quad k \text{ successes}, n-k \text{ failures} \\ \text{Posterior}: &\quad p|X \sim \text{Beta}(\alpha_0 + k, \beta_0 + n - k) \end{aligned}$$

The posterior is just the prior with successes and failures added!

11.3 Conjugate Priors

When the prior and posterior belong to the **same family of distributions**, the prior is called a **conjugate prior** for that likelihood.

The Beta distribution is the conjugate prior for the Bernoulli/Binomial likelihood. This is mathematically convenient—the posterior has a known, closed-form distribution.

Example:

Coin Flipping **Prior**: I have no strong belief, so I use Beta$(1, 1)$ (uniform).

$$E[p] = \frac{1}{2} = 0.5$$

Data: I flip the coin 10 times and get 7 heads, 3 tails.
Posterior: Beta$(1 + 7, 1 + 3) =$ Beta$(8, 4)$

$$E[p|\text{data}] = \frac{8}{12} = 0.67$$

My belief shifted from 0.5 to 0.67 based on the evidence!

12 Preview: Hierarchical Models

What if we have **grouped data**? For example, estimating shooting percentages for multiple NBA players?

12.1 The Problem

Option 1 (Pooled): Assume all players have the same p. Too restrictive—LeBron James is different from a rookie.

Option 2 (Separate): Estimate each player's p independently. Problem: A rookie with 5 shots has very uncertain estimate.

Option 3 (Hierarchical): The best of both worlds!

12.2 The Hierarchical Approach

1. **Level 1 (Data)**: Each player j's shots follow their own probability p_j
2. **Level 2 (Players)**: The p_j's themselves come from a common distribution

$$p_j \sim \text{Beta}(\alpha, \beta)$$

3. **Level 3 (League)**: The hyperparameters α, β might have their own prior (hyperprior)

Key Information

Why Hierarchical Models Work

Hierarchical models **share information** across groups.

- LeBron James (1000 shots): His p_j is mostly determined by his own data
- Rookie (10 shots): His p_j is **shrunk toward the league average**, borrowing strength from other players

This "shrinkage" produces better estimates for players with limited data!

Warning

Bayesian Logistic Regression: Not Beta!

Can we use Beta as a prior for logistic regression? **No!**

- Beta is for probabilities $p \in [0, 1]$
- Logistic regression parameters β can be any real number $(-\infty, \infty)$

For logistic regression, we typically use **Normal distributions** as priors for β:

$$\beta_j \sim N(0, \sigma^2)$$

A prior centered at 0 means "I expect this coefficient to be small"—similar to Ridge regularization!

13 Summary

Multiclass Classification

- **Multinomial**: $K-1$ models comparing each class to a reference
- **OvR**: K separate "class vs others" classifiers
- **Softmax**: Converts scores to probabilities summing to 1
- **Prediction**: Choose class with highest probability (plurality)

Classification Evaluation

- **Confusion Matrix**: TP, FP, TN, FN
- **TPR (Sensitivity)**: $\frac{TP}{TP+FN}$ — catching positives
- **FPR**: $\frac{FP}{FP+TN}$ — false alarms
- **ROC Curve**: TPR vs FPR across all thresholds
- **AUC**: Area under ROC; $1 =$ perfect, $0.5 =$ random

Bayesian Inference

- **Bayes' Rule**: Posterior $\propto$ Likelihood $\times$ Prior
- **Beta Distribution**: Prior for probabilities, $p \in [0, 1]$
- **Beta-Binomial**: $\text{Beta}(\alpha, \beta) \to \text{Beta}(\alpha + k, \beta + n - k)$
- **Conjugate Prior**: Prior and posterior same family
- **Hierarchical Models**: Share information across groups

Key Formulas

Softmax: $P(Y = k) = \frac{e^{s_k}}{\sum_j e^{s_j}}$
TPR: $\frac{TP}{TP+FN}$ **FPR:** $\frac{FP}{FP+TN}$
Beta Mean: $E[X] = \frac{\alpha}{\alpha+\beta}$
Beta-Binomial Update: $\text{Beta}(\alpha_0 + k, \beta_0 + n - k)$