ML Advanced Topics 1

Class Imbalance in Machine Learning

Learning Objectives:

- Understand the class imbalance problem and its impact on model performance
- Master evaluation metrics for imbalanced datasets
- Learn classical and modern techniques to handle imbalance
- Apply cost-sensitive learning and advanced sampling methods

The Class Imbalance Problem

Definition:

Class imbalance occurs when one class significantly outnumbers other classes in the training dataset.

Real-World Examples:

- **Fraud detection:** 0.1% fraudulent transactions, 99.9% legitimate
- **Medical diagnosis:** 1-5% disease positive, 95-99% negative
- **Manufacturing defects:** 0.01-1% defective, 99-99.99% normal
- **Spam detection:** 10-20% spam, 80-90% legitimate emails
- **Anomaly detection:** <0.1% anomalies, >99.9% normal behavior

The Problem:

Standard ML algorithms assume balanced class distribution and optimize for overall

Why Is This a Problem?

Example: Fraud Detection

Dataset: 10,000 transactions (100 fraudulent, 9,900 legitimate)

Naive Classifier: Predict everything as "legitimate"

- **Accuracy:** 99% ✓ (looks great!)
- **Fraud detected:** 0% ✗ (complete failure!)

Mathematical Formulation:

Given training data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$

Imbalance ratio: $\rho = \frac{N_{\text{majority}}}{N_{\text{minority}}}$

For $\rho >> 1$ (e.g., $\rho = 100$), standard loss functions fail.

Impact on Learning

Gradient Dominance:

Standard cross-entropy loss:

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

With imbalance ($N_+ = 100, N_- = 9900$):

- Majority class contributes $99\times$ more to gradient updates
- Model learns to ignore minority class patterns
- Decision boundary shifts toward majority class

Key Insight: Optimization pressure overwhelmingly favors majority class.

Evaluation Metrics for Imbalanced Data

Confusion Matrix:

	Predicted Positive	**Predicted Negative**
Actual Positive	TP (True Positive)	FN (False Negative)
Actual Negative	FP (False Positive)	TN (True Negative)

Evaluation Metrics for Imbalanced Data

Critical Metrics:

Precision: Of predicted positives, how many are correct?

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall (Sensitivity): Of actual positives, how many did we find?

$$\text{Recall} = \frac{TP}{TP + FN}$$

Advanced Evaluation Metrics

F1-Score: Harmonic mean of precision and recall

$$F_1 = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2TP}{2TP + FP + FN}$$

F-Beta Score: Weighted harmonic mean

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{Precision} \times \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}}$$

- $\beta > 1$: Favor recall (medical diagnosis)
- $\beta < 1$: Favor precision (spam detection)

Advanced Evaluation Metrics

Matthews Correlation Coefficient (MCC):

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Range: $[-1, 1]$, balanced metric even for extreme imbalance

ROC and PR Curves

ROC Curve (Receiver Operating Characteristic):

- X-axis: False Positive Rate = $\frac{FP}{FP+TN}$
- Y-axis: True Positive Rate (Recall) = $\frac{TP}{TP+FN}$
- **AUC-ROC:** Area Under ROC Curve

Precision-Recall Curve:

- X-axis: Recall = $\frac{TP}{TP+FN}$
- Y-axis: Precision = $\frac{TP}{TP+FP}$
- **AUC-PR:** Area Under PR Curve

Important: For imbalanced data, **PR curves are more informative** than ROC curves!

ROC can be overly optimistic due to large TN count.

Approach 1: Data-Level Methods - Resampling

1. Random Oversampling:

Randomly duplicate minority class samples

$$\mathcal{D}_{\text{new}} = \mathcal{D}_{\text{maj}} \cup \mathcal{D}_{\text{min}} \cup \text{sample}(\mathcal{D}_{\text{min}}, k)$$

Pros: Simple, no information loss

Cons: Overfitting (exact duplicates), no new information

2. Random Undersampling:

Randomly remove majority class samples

$$\mathcal{D}_{\text{new}} = \text{sample}(\mathcal{D}_{\text{maj}}, n) \cup \mathcal{D}_{\text{min}}$$

Pros: Faster training, balanced classes

Cons: Information loss, reduced dataset size

SMOTE: Synthetic Minority Over-sampling Technique

Algorithm (Chawla et al., 2002):

1. For each minority sample $\mathbf{x}_i$:
   - Find k nearest neighbors in minority class: $\mathcal{N}_k(\mathbf{x}_i)$
   - Select random neighbor $\mathbf{x}_j \in \mathcal{N}_k(\mathbf{x}_i)$
   - Generate synthetic sample:

$$\mathbf{x}_{\text{new}} = \mathbf{x}_i + \lambda(\mathbf{x}_j - \mathbf{x}_i), \quad \lambda \sim \text{Uniform}(0, 1)$$

Key Innovation: Creates synthetic samples along the line segments connecting minority class neighbors.

SMOTE: Synthetic Minority Over-sampling Technique

Advantages:

- Generalizes minority class distribution
- Reduces overfitting compared to random oversampling
- Widely used baseline

Limitations:

- Can generate noisy samples in overlap regions
- Doesn't consider majority class distribution

Advanced Sampling: ADASYN

ADASYN (Adaptive Synthetic Sampling, 2008):

Focus synthetic sample generation on harder-to-learn minority regions.

Algorithm:

1. Calculate density distribution Γ_i for each minority sample:

$$\Gamma_i = \frac{\Delta_i}{k}$$

where Δ_i = number of majority samples in k-NN

2. Normalize: $\hat{\Gamma}_i = \Gamma_i / \sum_{i=1}^{m} \Gamma_i$

3. Calculate number of synthetic samples for each $\mathbf{x}_i$:

$$g_i = \hat{\Gamma}_i \times G, \quad G = (N_{\text{maj}} - N_{\text{min}})$$

Advanced Sampling: ADASYN

4. Generate g_i samples around $\mathbf{x}_i$ using SMOTE-like interpolation

Key Insight: Harder-to-learn samples (surrounded by majority) get more synthetic neighbors.

Borderline-SMOTE

Motivation: Only generate synthetic samples near decision boundary.

Algorithm (Han et al., 2005):

1. For each minority sample $\mathbf{x}_i$, find k-NN
2. Count majority samples in k-NN: m
3. Classify minority samples:
   - **DANGER:** $\frac{k}{2} \leq m < k$ (borderline samples)
   - **SAFE:** $m < \frac{k}{2}$ (well-separated)
   - **NOISE:** $m = k$ (surrounded by majority)
4. Generate synthetic samples only for **DANGER** samples using SMOTE

Borderline-SMOTE

Variants:

- **Borderline-SMOTE1:** Interpolate only between minority samples
- **Borderline-SMOTE2:** Interpolate between minority and nearest majority

Advantage: Focuses augmentation where it matters most (decision boundary).

Approach 2: Algorithm-Level Methods

Cost-Sensitive Learning

Idea: Assign higher misclassification cost to minority class.

Modified Loss Function:

$$\mathcal{L}_{\text{weighted}} = -\sum_{i=1}^{N} w_i \left[y_i \log(p_i) + (1 - y_i) \log(1 - p_i) \right]$$

Where:

$$w_i = \begin{cases} w_{\text{min}} & \text{if } y_i = 1 \text{ (minority)} \\ w_{\text{maj}} & \text{if } y_i = 0 \text{ (majority)} \end{cases}$$

Approach 2: Algorithm-Level Methods

Cost-Sensitive Learning

Common Weighting Schemes:

1. **Inverse Frequency:** $w_c = \frac{N}{N_c}$
2. **Effective Number:** $w_c = \frac{1-\beta}{1-\beta^{N_c}}, \beta \in [0, 1)$
3. **Square Root:** $w_c = \frac{1}{\sqrt{N_c}}$

Focal Loss (Lin et al., 2017)

Problem with Cross-Entropy:

Easy-to-classify samples (majority) dominate loss, even with weighting.

Focal Loss:

$$\mathcal{L}_{\text{focal}} = -\alpha(1 - p_t)^{\gamma} \log(p_t)$$

Where:

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{if } y = 0 \end{cases}$$

Parameters:

- $\alpha \in [0, 1]$: Class weight (typically $\alpha = 0.25$)
- $\gamma \geq 0$: Focusing parameter (typically $\gamma = 2$)

Focal Loss Analysis

Effect of Focusing Parameter γ:

- $\gamma = 0$: Standard cross-entropy
- $\gamma = 1$: Moderate down-weighting
- $\gamma = 2$: Strong focus on hard examples (common choice)
- $\gamma = 5$: Extreme focusing

Example:

For $\gamma = 2$:

- $p_t = 0.9$ (easy): Weight factor = $(1 - 0.9)^2 = 0.01$
- $p_t = 0.5$ (hard): Weight factor = $(1 - 0.5)^2 = 0.25$
- $p_t = 0.1$ (very hard): Weight factor = $(1 - 0.1)^2 = 0.81$

Focal Loss Analysis

Applications:

- Object detection (RetinaNet)
- Dense prediction tasks
- Highly imbalanced classification

Ensemble Methods for Imbalanced Data

1. Balanced Random Forest (Chen et al., 2004):

- Bootstrap minority class completely
- Randomly undersample majority class
- Build decision tree on balanced bootstrap
- Repeat for all trees

2. EasyEnsemble (Liu et al., 2009):

- Create multiple balanced subsets via random undersampling
- Train classifier on each subset
- Combine via voting/averaging
- Maintains diversity through different majority subsamples

RUSBoost Algorithm (Seiffert et al., 2010)

Combines: Random Undersampling + AdaBoost

Algorithm:

For $t = 1$ to T:

1. Create balanced training set via random undersampling:

$$\mathcal{D}_t^{\text{balanced}} = \text{RUS}(\mathcal{D}, w_t)$$

2. Train weak learner h_t on $\mathcal{D}_t^{\text{balanced}}$
3. Calculate error: $\epsilon_t = \sum_{i:h_t(\mathbf{x}_i) \neq y_i} w_t^{(i)}$
4. Calculate learner weight: $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$

RUSBoost Algorithm (Seiffert et al., 2010)

Combines: Random Undersampling + AdaBoost

5. Update sample weights:

$$w_{t+1}^{(i)} = w_t^{(i)} \exp(-\alpha_t y_i h_t(\mathbf{x}_i))$$

Final prediction: $H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$

Class-Balanced Loss (Cui et al., 2019)

Effective Number of Samples:

As samples increase, marginal benefit decreases (diminishing returns).

Definition:

$$E_n = \frac{1 - \beta^n}{1 - \beta}$$

where $\beta \in [0, 1)$ controls the diminishing rate.

Class-Balanced Loss:

$$\mathcal{L}_{\text{CB}} = \frac{1}{E_n} \mathcal{L}$$

Class-Balanced Loss (Cui et al., 2019)

Re-weighting:

$$w_c = \frac{1-\beta}{1-\beta^{N_c}}$$

Combining with Focal Loss:

$$\mathcal{L}_{\text{CB-Focal}} = \frac{1-\beta}{1-\beta^{N_c}} \cdot (-\alpha(1-p_t)^{\gamma}\log(p_t))$$

Deep Learning: Long-Tailed Recognition

Problem: Real-world data follows long-tailed distribution

- Few classes with many samples (head)
- Many classes with few samples (tail)

Recent Approaches:

1. Decoupling (Kang et al., 2020):

- **Stage 1:** Train representation with instance-balanced sampling
- **Stage 2:** Fine-tune classifier with class-balanced sampling
- Key insight: Good representations learned from all data; classifier needs balancing

Deep Learning: Long-Tailed Recognition

2. Balanced Softmax (Ren et al., 2020):

$$p(y = c|\mathbf{x}) = \frac{\exp(f_c(\mathbf{x}) + \log(\pi_c))}{\sum_{j=1}^{C} \exp(f_j(\mathbf{x}) + \log(\pi_j))}$$

where $\pi_c = N_c/N$ is class prior

Metric Learning for Imbalanced Data

Large Margin Cosine Loss (LMCL, Wang et al., 2018):

$$\mathcal{L}_{\text{LMCL}} = -\log \frac{e^{s(\cos(\theta_{y_i})-m)}}{e^{s(\cos(\theta_{y_i})-m)} + \sum_{j \neq y_i} e^{s\cos(\theta_j)}}$$

where:

- θ_j is angle between feature and class center j
- s: scaling factor
- m: margin parameter

Advantage: Enforces larger margins for minority classes, improving generalization.

Metric Learning for Imbalanced Data

Class-Balanced Variant:

$$m_c = m_0 \cdot \left(\frac{N_{\max}}{N_c} \right)^{\alpha}$$

Smaller classes get larger margins.

Self-Supervised Learning for Imbalance

Contrastive Learning (SimCLR, MoCo):

1. Learn representations without labels via contrastive loss:

$$\mathcal{L}_{\text{contrastive}} = -\log \frac{\exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_i^+)/\tau)}{\sum_k \exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_k)/\tau)}$$

2. Fine-tune on imbalanced labeled data

Advantages:

- Pre-training uses all data equally (no class imbalance)
- Learns robust features before classification
- Particularly effective for extremely imbalanced scenarios

Recent Work: Self-supervised + class-balanced fine-tuning achieves SOTA on long-tailed

Foundation Models & Transfer Learning (2024-2025)

Vision-Language Models (CLIP, ALIGN):

- Pre-trained on billions of image-text pairs
- Transfer learning mitigates imbalance in downstream tasks
- **Fine-tuning strategy:** Resample minority classes during adaptation

Recent Findings (2024):

- CLIP fine-tuning with class resampling improves minority detection
- Foundation models learn robust features less affected by imbalance
- Few-shot learning particularly effective for rare classes

Application: Construction safety violation detection (2024)

- Fine-tuned CLIP on imbalanced safety violation data

Advanced Deep Generative Methods (2024)

Beyond GANs - Modern Generative Approaches:

1. Diffusion-based Oversampling:

- Use diffusion models (DDPM) for minority class generation
- Better mode coverage than GANs
- More stable training

2. VAE Variants:

- **Conditional VAE (CVAE):** Class-conditional generation
- **Improved VAEGAN:** Combines VAE encoder with GAN discriminator
- **ARAE:** Adversarially Regularized Autoencoder

3. Specialized GANs (2024 Survey):

Meta-Learning Approaches

MLDG for Imbalance (MetaSAug, Li et al., 2021):

Meta-learn how to augment minority class data.

Algorithm:

1. Split training data into meta-train and meta-val
2. Learn augmentation parameters ϕ via meta-optimization:

$$\min_{\phi} \mathcal{L}_{\text{val}}\left(f_{\theta^*(\phi)}\right)$$

where $\theta^*(\phi) = \arg\min_\theta \mathcal{L}_{\text{train}}(\theta, \phi)$

3. Apply learned augmentation to minority class

Key Insight: Meta-learning discovers augmentations that improve validation performance on imbalanced distribution.

GAN-based Oversampling

BAGAN (Balanced GAN, Mariani et al., 2018):

1. Train GAN to generate minority class samples:
   - Generator: $G(z, c) \rightarrow \mathbf{x}$
   - Discriminator: $D(\mathbf{x}) \rightarrow (\text{real/fake}, c)$
2. Auto-encoder ensures realistic samples:

$$\mathcal{L}_{\text{autoencoder}} = \|\mathbf{x} - \text{Decoder}(\text{Encoder}(\mathbf{x}))\|^2$$

3. Class balancing via class-conditional generation

GAN-based Oversampling

Advantages:

- Generates diverse, realistic samples
- Can capture complex minority class distributions

Challenges:

- Training instability
- Mode collapse
- Computational cost

Curriculum Learning for Imbalance

Strategy: Gradually transition from balanced to imbalanced distribution.

Dynamic Sampling (Pouyanfar et al., 2018):

Epoch t sampling probability for class c:

$$p_c^{(t)} = \frac{(1 - \alpha_t) \cdot \frac{1}{C} + \alpha_t \cdot \frac{N_c}{N}}{\sum_{j=1}^{C} \left[(1 - \alpha_t) \cdot \frac{1}{C} + \alpha_t \cdot \frac{N_j}{N} \right]}$$

where $\alpha_t = \frac{t}{T}$ (increases from 0 to 1)

Curriculum Learning for Imbalance

Effect:

- Early training: Balanced sampling ($\alpha_t \approx 0$)
- Late training: Natural distribution ($\alpha_t \approx 1$)

Rationale: Learn robust features early, adapt to true distribution later.

Hybrid Approaches

Combining Multiple Techniques:

1. SMOTE + Ensemble:

- Apply SMOTE to balance data
- Train ensemble (Random Forest, XGBoost)
- Combines data-level and algorithm-level methods

2. Cost-Sensitive + Focal Loss:

$$\mathcal{L} = w_c \cdot (-\alpha(1 - p_t)^\gamma \log(p_t))$$

- Double down-weighting: class weights + focal term

Hybrid Approaches

3. Two-Stage Training:

- Stage 1: Self-supervised pre-training on full data
- Stage 2: Fine-tune with class-balanced loss + data augmentation

Best Practice: Combine techniques based on dataset characteristics and domain constraints.

Practical Guidelines

When to Use What:

Imbalance Ratio	Recommended Approach
2:1 to 5:1	Class weights, stratified sampling
5:1 to 20:1	SMOTE, cost-sensitive learning
20:1 to 100:1	ADASYN, focal loss, ensemble methods
>100:1	Anomaly detection, one-class learning

Practical Guidelines

Domain-Specific Considerations:

- **Medical:** High recall priority → F2 score, asymmetric costs
- **Fraud:** Real-time constraints → efficient methods (undersampling)
- **Manufacturing:** Interpretability → rule-based ensembles

Implementation Tips

1. Always Use Stratified Splitting:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, stratify=y, test_size=0.2
)
```

2. Combine Techniques Carefully:

- SMOTE on training set only (prevent data leakage)
- Apply after train-test split
- Don't combine SMOTE + random oversampling (redundant)

Implementation Tips

3. Monitor Multiple Metrics:

- Precision, Recall, F1 for each class
- Confusion matrix
- PR-AUC (more informative than ROC-AUC)

4. Threshold Tuning:

Adjust decision threshold based on business costs, not default 0.5.