Imbalanced Classification — Practical Methods

A concise overview of four core techniques to handle class imbalance:

- Random Oversampling
- Synthetic Minority Oversampling (SMOTE)
- Random Undersampling
- lass-Weighted/Loss-Adjusted training.

When to use

- Positive class is rare (e.g., fraud, defects, disease).
- Baseline classifier biased toward majority predictions.
- Metrics like accuracy are misleading; use ROC-AUC/PR-AUC, recall, F_β.
- $\beta < 1$ gives more emphasis on Precision. $\beta > 1$ gives more emphasis on Recall.

At a glance

Using these methods should improve recall (sensitivity) for the minority class

Calibrated Loss **Robust CV**

Always evaluate with stratified splits and metrics sensitive to imbalance; consider threshold tuning post-training.

1) Random Oversampling

Duplicate minority samples until class counts are balanced (or closer to balanced).

Simple example

Dataset: 1,000 samples with 50 positive (minority) and 950 negative (majority). Oversample positives by duplicating them ~19× to reach ≈950 positives. Train the model on the augmented dataset.

> **Pros:** Easy; preserves minority distribution; works with any model.
> **Cons:** Risk of overfitting (exact duplicates); no new information introduced.

Practical notes

- Combine with data augmentation (noise, jitter) for robust models.
- Apply inside cross-validation folds to avoid leakage.
- Useful baseline to compare against SMOTE variants.

2) SMOTE — Synthetic Minority Oversampling Technique

Generate synthetic minority samples by interpolating between a sample and one of its minority nearest neighbors.

Description (with tiny example)

Pick a minority point $\mathbf{x}_i$. Find its k-nearest minority neighbors. Randomly choose one neighbor $\mathbf{x}_z$. Draw $\lambda \sim \mathcal{U}(0,1)$ and create:

$x_{new} = x_i + \lambda(x_z - x_i)$

Example (1D intuition): if $x_i = 2.0$ and neighbor $x_z = 5.0$, a sample with $\lambda = 0.3$ yields $x_{\text{new}} = 2 + 0.3(5-2) = 2.9$.

> **Pros:** Reduces overfitting vs pure duplication; smooths decision regions.
> **Cons:** Can create ambiguous samples near class boundaries; mind feature scaling & categorical features.

Mathematical description

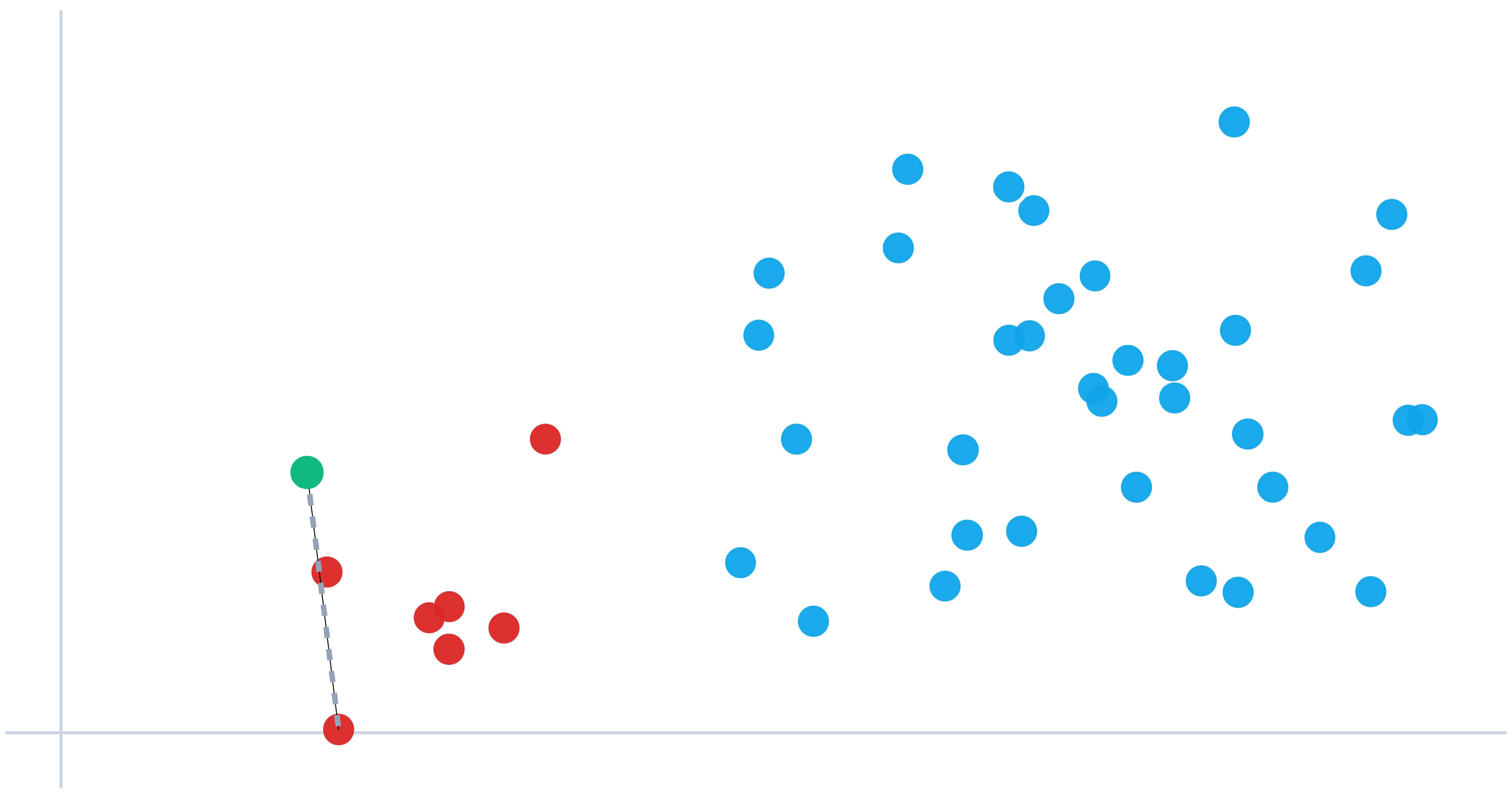
Given minority set $\mathcal{X}_{\min} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$, choose $\mathbf{x}_i \in \mathcal{X}_{\min}$, pick neighbor $\mathbf{x}_z \in \text{NN}_k(\mathbf{x}_i) \cap \mathcal{X}_{\min}$, and sample

$$\mathbf{x}_{\text{new}} \;=\; \mathbf{x}_i \;+\; \lambda\big(\mathbf{x}_z - \mathbf{x}_i\big), \qquad \lambda \sim \mathcal{U}(0,1).$$

Repeat until desired oversampling ratio is met. Use distances after proper normalization; categorical features require specialized variants (e.g., SMOTE-NC).

Interactive SMOTE Animation

A minority point (red) picks a red neighbor; the green dot shows a synthetic sample sliding along the segment $\mathbf{x}_i \to \mathbf{x}_z$ as $\lambda \in [0, 1]$.



Randomize points Play λ: 0.00 Majority Minority Synthetic

3) Random Undersampling

Remove a subset of majority samples to reduce imbalance (and training time).

Simple example

Dataset: 100 positive (minority) vs 9,900 negative (majority).
Randomly discard ~9,800 majority samples to reach ~100 vs 100.
Train on the reduced set.

> **Pros:** Faster training; simplifies decision boundary; useful when majority is very redundant.
> **Cons:** Risk of discarding informative majority cases; higher variance if too aggressive.

Practical notes

- Prefer stratified undersampling or informed heuristics (e.g., keep “hard negatives”).
- Combine with cost-sensitive loss instead of extreme undersampling.
- Always apply inside CV folds to avoid leakage.

4) Class Weights / Loss Function Adjustment

Modify the loss so that mistakes on the minority class cost more than those on the majority class.

Binary cross-entropy with class weights

Let $y_i \in \{0, 1\}$, prediction $p_i = \Pr(y_i = 1 \mid \mathbf{x}_i)$, and weights w_1 (for positives) and w_0 (for negatives). The weighted loss is

$$\mathcal{L}_{\text{wBCE}} = -\frac{1}{N} \sum_{i=1}^{N} \Big(w_1 \, y_i \log p_i \; + \; w_0 \, (1 - y_i) \log(1 - p_i) \Big).$$

Common choice (inverse frequency): for class $c \in \{0, 1\}$ with count n_c, total n, and $C = 2$ classes,

$$w_c \; = \; \frac{n}{C \, n_c}.$$

Larger w_1 increases the gradient for positive errors, improving recall but potentially reducing precision. Tune threshold after training.

Margin-based models (SVM, etc.)

For a linear SVM with hinge loss, class-specific penalty C_c scales the slack term:

$$\min_{\mathbf{w}, b, \boldsymbol{\xi}} \; \frac{1}{2} \|\mathbf{w}\|^2 \; + \; \sum_{i=1}^{N} C_{y_i} \, \xi_i \quad \text{s.t.} \quad y_i \left(\mathbf{w}^\top \mathbf{x}_i + b \right) \geq 1 - \xi_i, \; \xi_i \geq 0.$$

Setting $C_1 > C_0$ makes positive-class violations more costly.

Simple example

Dataset: 200 positives, 4,800 negatives ($n = 5{,}000$). Using $w_c = \frac{n}{C n_c}$ with $C = 2$:

- $w_1 = \frac{5000}{2 \cdot 200} = 12.5$
- $w_0 = \frac{5000}{2 \cdot 4800} \approx 0.5208$

These weights push the optimizer to focus more on minority errors.

Quick Comparison & Tips

Method comparison

- **Random Oversampling:** Baseline; risk of overfitting; consider light noise.
- **SMOTE:** Better generalization; take care near class boundaries; scale features.
- **Random Undersampling:** Faster; may lose signal; pair with cost-sensitive loss.
- **Class Weights:** Model-native; no data duplication; pair with threshold tuning.

Evaluation checklist

- Use *stratified* CV and maintain time order for temporal data.
- Prefer PR-AUC and recall @ target precision for rare positives.
- Tune decision threshold on a validation set aligned to business costs.
- Guard against leakage: perform resampling *inside* each CV fold.

Class Weight parameters of popular classification models

Model	Parameter	Options	Default
LogisticRegression	class_weight	None, "balanced", dict	None
DecisionTreeClassifier	class_weight	None, "balanced", dict	None
RandomForestClassifier	class_weight	None, "balanced", "balanced_subsample", dict	None
XGBClassifier	scale_pos_weight	Positive float (e.g., n_{neg}/n_{pos})	1

Meaning of `"balanced"` and `"balanced_subsample"`

For class c with n_c samples in a dataset of total size n and C classes:

$$w_c = \frac{n}{C \cdot n_c}$$

- **"balanced":** The weights w_c are computed using the whole training set once, then applied globally.
- **"balanced_subsample":** For ensembles (e.g. Random Forests), the same formula is applied but on each bootstrap sample drawn for a tree, so n and n_c are computed *per-subsample*.