Clase 064 — Class imbalance: SMOTE, ADASYN, class_weight, threshold tuning

Parte: 1 — Machine Learning Clásico · Fuente: Chawla et al. (2002) SMOTE + He et al. (2008) ADASYN + imbalanced-learn docs. Duración estimada: 80 min.

Clase 064 — Class imbalance: SMOTE, ADASYN, class_weight, threshold tuning

> *Parte: 1 — Machine Learning Clásico · Fuente: Chawla et al. (2002) SMOTE + He et al. (2008) ADASYN + imbalanced-learn docs. Duración estimada: 80 min.*

Objetivo

Tratar datasets desbalanceados —fraude (1 % positivo), churn (5 %), enfermedades raras—. Las trampas son sutiles: accuracy puede ser 99 % con un clasificador trivial. Cubrir las 4 estrategias estándar: class_weight, threshold tuning, oversampling (SMOTE, ADASYN), undersampling (Tomek, ENN). Y la decisión clave: ¿qué métrica reportar? (F1, PR-AUC, MCC, no accuracy).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Detectar imbalance: value_counts(normalize=True). Decidir si > 10:1 amerita tratamiento.
- Aplicar class_weight='balanced' o pesos custom en sklearn.
- Aplicar threshold tuning: optimizar el umbral de decisión sobre la curva PR según la métrica del negocio.
- Usar SMOTE (synthetic minority over-sampling) de imbalanced-learn: SMOTE(k_neighbors=5).fit_resample(X, y).
- Combinar oversampling + undersampling (SMOTETomek, SMOTEENN).
- Reportar PR-AUC y MCC (Matthews Correlation Coefficient) en lugar de accuracy.

Temas

- Imbalance ratio: >10:1 problemático.
- Métricas: precision, recall, F1, F-beta, PR-AUC, MCC.
- class_weight: penalizar más errores en minoría durante training.
- Threshold tuning: mover el umbral fuera del 0.5 default.
- SMOTE: interpola entre vecinos de la minoría.
- ADASIN: como SMOTE pero con más densidad en zonas "difíciles".
- Tomek links / ENN: remueve borderline de la mayoría.
- imbalanced-learn pipelines.

Definiciones y características

- Class imbalance: una clase mucho más frecuente que otra(s).
- class_weight='balanced': pesos automáticos inversamente proporcionales a frecuencia.
- SMOTE: para cada sample minoritario, crear sintéticos en línea recta a sus k vecinos.
- ADASIN: SMOTE adaptativo — genera más cerca de la frontera de decisión.
- PR-AUC: área bajo Precision-Recall curve. Más informativa que ROC-AUC cuando hay imbalance fuerte.
- MCC: (TPTN - FPFN) / sqrt(...). Único valor en [-1, 1]. Robusto a imbalance.

- Threshold tuning: elegir el cutoff p > τ → predict 1 que maximiza la métrica de negocio.

Dataset / recursos

- fetch_openml('creditcardfraud') (Kaggle, 0.17 % positivo).
- Librerías: imbalanced-learn (pip install imbalanced-learn), scikit-learn.

Ejercicios

1. Baseline sin tratamiento: LogisticRegression en creditcardfraud. Accuracy alto, recall pésimo.
2. class_weight: LogisticRegression(class_weight='balanced'). Recall sube, precision baja.
3. Threshold tuning: con probabilidades de predict_proba, barrer thresholds y plotear F1 vs threshold. Elegir el óptimo.
4. SMOTE: from imblearn.over_sampling import SMOTE; X_res, y_res = SMOTE().fit_resample(X_train, y_train). Entrenar y evaluar.
5. Pipeline imblearn: Pipeline([('smote', SMOTE()), ('clf', LogisticRegression())]). Importante: SMOTE solo se aplica en train (imblearn pipeline lo maneja).

Homework verificable

Sobre creditcardfraud:

1. 4 modelos: baseline, class_weight, SMOTE, SMOTETomek.
2. Reportar precision, recall, F1, PR-AUC y MCC para cada uno.
3. Curva PR de los 4 lado a lado.
4. Threshold tuning sobre el mejor para maximizar F2 (favorece recall).

Criterio de aceptación: al menos uno de los tratamientos supera al baseline en F1 por ≥ 0.1; PR-AUC ≥ 0.8.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Accuracy 99.5 % en fraude con clasificador	Reporta accuracy en imbalance. Fix: usar P
SMOTE aplicado antes de split	Leakage: samples sintéticos derivados de t
Threshold default 0.5 con probabilidades c	Subóptimo. Fix: tunear sobre val.
SMOTE con features categóricas codificadas	Interpola entre 0/1, sin sentido. Fix: SMO
Oversampling para clase de 0.1 % a 50/50	Excesivo. Fix: ratio 1:3 o 1:5 suele ser s

Preguntas frecuentes

class_weight o SMOTE?

class_weight es más simple y barato. SMOTE puede ayudar en casos extremos o con árboles/ensembles. Probá ambos.

¿Cuándo PR-AUC vs ROC-AUC?

PR-AUC cuando imbalance fuerte (>10:1) — más sensible. ROC-AUC para casos balanceados o solo para comparar relativamente.

¿MCC vs F1?

MCC es simétrico (trata clases igual). F1 favorece la minoría. Para fraude/medical, MCC es más conservador.

¿SMOTE con DL?

Menos común — DL prefiere ajustar la loss (focal loss, weighted CE).

¿Undersampling pierde información?

Sí. Por eso es un last resort. SMOTE/oversampling sintético suele ser mejor.

Referencias

- Chawla et al. (2002), SMOTE, JAIR.
- He et al. (2008), ADASYN, IJCNN.
- Saito & Rehmsmeier (2015), The Precision-Recall Plot Is More Informative than the ROC Plot, PLOS ONE.
- imbalanced-learn docs.

Siguiente clase

Clase 065 — Precision/Recall tradeoff

Apéndice: notebook (primer bloque)

Binary sintético con ratio 1:99. Comparamos baseline vs class_weight vs SMOTE vs ADASYN, en métricas relevantes (PR-AUC, recall, no accuracy). Instalar: pip install imbalanced-learn.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.metrics import (accuracy_score, recall_score, precision_score,
                             average_precision_score, precision_recall_curve)

try:
    from imblearn.over_sampling import SMOTE, ADASYN
    from imblearn.pipeline import Pipeline as ImbPipeline
    IMBLEARN_OK = True
except ImportError:
    print('instalar: pip install imbalanced-learn')
    IMBLEARN_OK = False

np.random.seed(42)
```

Archivos complementarios

- notebook.ipynb