STATISTICS & MACHINE LEARNING · LESSON 21 OF 28

# Imbalanced Classification: Fraud Detection with Class Weights, Resampling and Thresholds

Most valuable classification problems are rare-event problems (fraud, defaults, churn, failures). Learn why accuracy and naive training fail, the right metrics (PR-AUC, recall at fixed precision, cost), class weights, undersampling, oversampling and SMOTE inside pipelines, threshold moving, and anomaly-detection alternatives, on a synthetic card-fraud dataset with about 1.4% fraud.

ML track (added) · scikit-learn 1.7+ · imbalanced-learn 0.1x · Advanced · 110 min

**By the end of this lesson you can**

- Recognise imbalanced problems and the metrics that work for them
- Apply class_weight and scale_pos_weight to make models pay attention to the rare class
- Use random undersampling, oversampling and SMOTE correctly inside cross-validation
- Move the decision threshold to meet a business constraint (alert capacity, cost, minimum precision)
- Use Isolation Forest when labels are scarce

# A rare-event problem

Python

```
import numpy as np, pandas as pd
rng = np.random.default_rng(7); n = 20000
df = pd.DataFrame({
    "amount": np.round(rng.lognormal(7, 1.1, n), 0),                    # rupees
    "hour": rng.integers(0, 24, n),
    "km_from_home": np.round(rng.exponential(8, n), 1),
    "new_merchant": rng.random(n) < 0.15,
    "online": rng.random(n) < 0.45,
    "txns_last_24h": rng.poisson(2, n),
    "card_age_days": rng.integers(10, 3000, n),
})
night = df.hour.isin([0, 1, 2, 3, 4])
score = (-8.4 + 1.35 * np.log1p(df.amount / 1000) + 1.5 * night + 0.075 * df.km_from_home +
1.5 * df.new_merchant + 0.75 * df.online
         + 0.5 * df.txns_last_24h - 0.00135 * df.card_age_days + 3.3 * (night & (df.online
== 1) & (df.new_merchant == 1))      # risky combination
         + 2.2 * (df.txns_last_24h >= 6) + rng.normal(0, .4, n))
df["fraud"] = (rng.random(n) < 1 / (1 + np.exp(-score))).astype(int)
df[["new_merchant", "online"]] = df[["new_merchant", "online"]].astype(int)
print(df.shape, f"fraud rate {df.fraud.mean():.2%} ({df.fraud.sum()} frauds)")
print(df.groupby("fraud")[["amount", "km_from_home", "new_merchant", "online",
"txns_last_24h", "card_age_days"]].median())
```

OUTPUT

```
(20000, 8) fraud rate 1.38% (276 frauds)
       amount  km_from_home  new_merchant  online  txns_last_24h  card_age_days
fraud
0      1074.5          5.50           0.0     0.0            2.0         1524.0
1      2118.0          8.85           1.0     1.0            3.0          744.5
```

With about 1.4% fraud, a model that **never** flags fraud is 98.6% accurate and completely useless.

## The right metrics

Python

```
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.metrics import accuracy_score, recall_score, precision_score, roc_auc_score,
average_precision_score, confusion_matrix
X, y = df.drop(columns="fraud"), df.fraud
X["log_amount"] = np.log1p(X.amount); X["night"] = X.hour.isin([0, 1, 2, 3, 4]).astype(int)
Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=0.3, stratify=y, random_state=0)
def report(name, model, t=0.5):
    p = model.predict_proba(Xte)[:, 1]; pred = p ⩾ t
    print(f"{name:34s} acc {accuracy_score(yte, pred):.3f} | recall {recall_score(yte,
pred):.3f} | precision {precision_score(yte, pred, zero_division=0):.3f} | ROC-AUC
{roc_auc_score(yte, p):.3f} | PR-AUC {average_precision_score(yte, p):.3f}")
    return p
plain = make_pipeline(StandardScaler(), LogisticRegression(max_iter=2000)).fit(Xtr, ytr)
p_plain = report("logistic (no adjustment)", plain)
print("random baseline PR-AUC = fraud rate =", round(yte.mean(), 3))
```

OUTPUT

```
logistic (no adjustment)           acc 0.986 | recall 0.072 | precision 0.545 | ROC-
AUC 0.942 | PR-AUC 0.321
random baseline PR-AUC = fraud rate = 0.014
```

| Use | Avoid |
| --- | --- |
| PR-AUC (average precision), recall at fixed precision, precision at k, expected cost | Accuracy |
| Confusion matrix at the chosen threshold | ROC-AUC alone (looks good even when precision is poor) |
| Stratified splits and CV | Random splits that leave few positives in validation |

## Fix 1: class weights

Tell the loss function that each fraud matters more: errors on the rare class are multiplied by its weight.

Python

```
weighted = make_pipeline(StandardScaler(), LogisticRegression(max_iter=2000,
class_weight="balanced")).fit(Xtr, ytr)
p_w = report("logistic, class_weight='balanced'", weighted)
print("balanced weights:", {0: round(float(len(ytr) / (2 * (ytr == 0).sum())), 2), 1:
round(float(len(ytr) / (2 * (ytr == 1).sum())), 2)})
```

OUTPUT

```
logistic, class_weight='balanced'  acc 0.867 | recall 0.904 | precision 0.087 | ROC-
AUC 0.944 | PR-AUC 0.274
balanced weights: {0: 0.51, 1: 36.27}
```

Weighting raises recall a lot (more frauds caught) at the price of precision (more false alarms). The **ranking** (AUC) hardly changes: weights mostly shift the probabilities, which is equivalent to moving the threshold. XGBoost and LightGBM use `scale_pos_weight` ≈ negatives ÷ positives.

## Fix 2: resampling

| Method | How | Risk |
|---|---|---|
| Random undersampling | Drop majority rows | Throws away information |
| Random oversampling | Duplicate minority rows | Overfitting to duplicates |
| SMOTE | Create synthetic minority rows by interpolating between neighbours | Synthetic points in wrong places; only for numeric features (SMOTENC for categorical) |

Python

```
from imblearn.pipeline import make_pipeline as imb_pipeline
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler, SMOTE
for name, sampler in [("undersample", RandomUnderSampler(random_state=0)), ("oversample",
RandomOverSampler(random_state=0)), ("SMOTE", SMOTE(random_state=0))]:
    m = imb_pipeline(StandardScaler(), sampler, LogisticRegression(max_iter=2000)).fit(Xtr,
ytr)
    _ = report(f"logistic + {name}", m)
```

OUTPUT

```
logistic + undersample             acc 0.869 | recall 0.892 | precision 0.087 | ROC-
AUC 0.939 | PR-AUC 0.237
```

```
logistic + oversample              acc 0.870 | recall 0.904 | precision 0.088 | ROC-
AUC 0.943 | PR-AUC 0.272
logistic + SMOTE                   acc 0.874 | recall 0.892 | precision 0.090 | ROC-
AUC 0.943 | PR-AUC 0.268
```

**WATCH OUT** **Resample only the training folds**

Resampling before the train/test split (or before cross-validation) puts copies or synthetic neighbours of test frauds into training: a classic leak. Use `imblearn.pipeline.Pipeline`, which resamples only during `fit`. Resampling also distorts predicted probabilities; recalibrate if you need true probabilities.

## Stronger model: gradient boosting

Python

```python
import lightgbm as lgb
pos_w = (ytr == 0).sum() / (ytr == 1).sum()
gbm = lgb.LGBMClassifier(n_estimators=400, learning_rate=0.03, num_leaves=15,
min_child_samples=50, subsample=0.8, subsample_freq=1, colsample_bytree=0.8, verbose=-1,
random_state=0).fit(Xtr, ytr)
p_g = report("LightGBM", gbm)
gbm_w = lgb.LGBMClassifier(n_estimators=400, learning_rate=0.03, num_leaves=15,
min_child_samples=50, scale_pos_weight=pos_w, verbose=-1, random_state=0).fit(Xtr, ytr)
_ = report("LightGBM, scale_pos_weight", gbm_w)
```

OUTPUT

```
LightGBM                           acc 0.987 | recall 0.169 | precision 0.560 | ROC-
AUC 0.947 | PR-AUC 0.346
LightGBM, scale_pos_weight         acc 0.963 | recall 0.651 | precision 0.218 | ROC-
AUC 0.937 | PR-AUC 0.348
```

## Fix 3: move the threshold (usually the best lever)

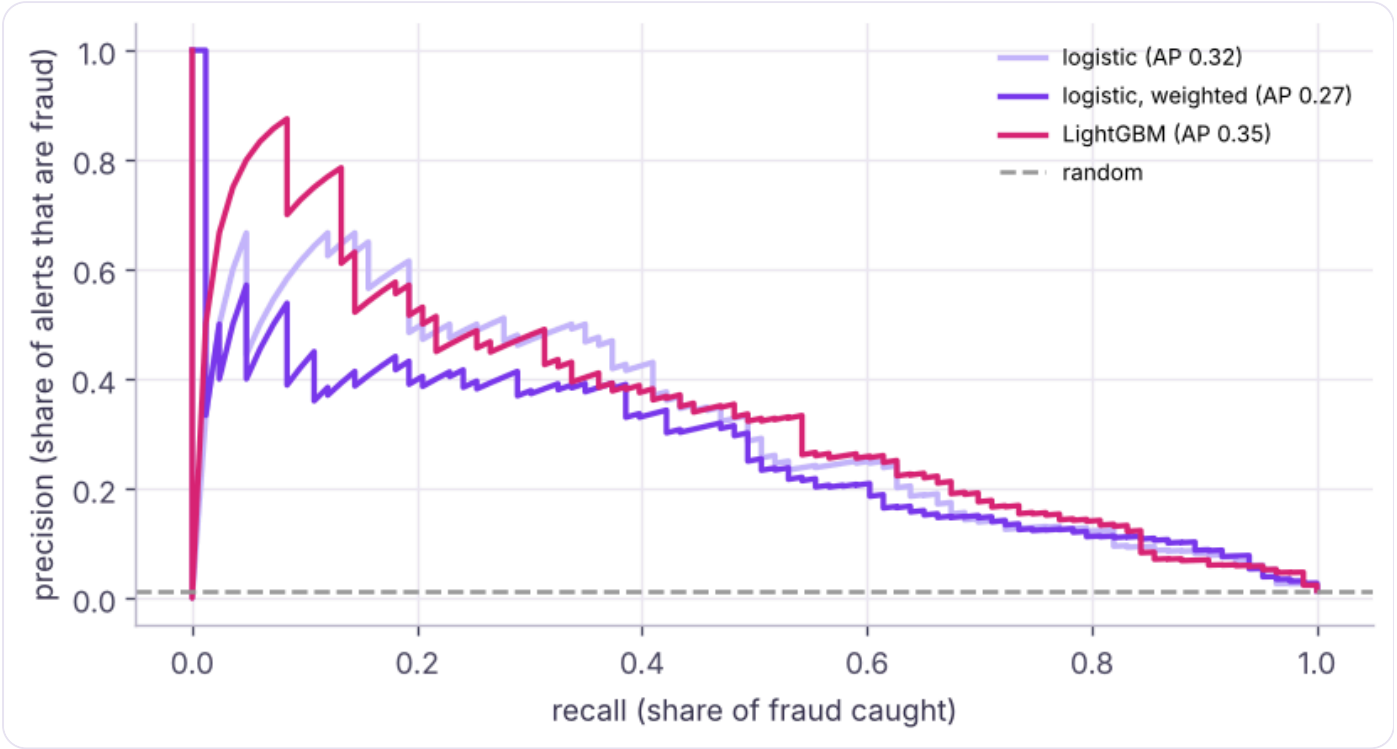
The fraud team can review **200 alerts a day** (out of about 6,000 test transactions per period here, scaled). Choose the threshold that fits the capacity, or the one that meets a minimum precision.

Python

```
from sklearn.metrics import precision_recall_curve
best_p = p_g
for k in (50, 100, 200, 400):
    t = np.sort(best_p)[-k]; pred = best_p ≥ t
    print(f"top {k:3d} alerts: threshold {t:.3f} | frauds caught {int((pred & (yte == 1)).sum()):3d} of {yte.sum()} (recall {recall_score(yte, pred):.2f}) | precision {precision_score(yte, pred):.2f}")
prec, rec, thr = precision_recall_curve(yte, best_p)
ok = np.where(prec[:-1] ≥ 0.5)[0]
if len(ok): print(f"highest recall with precision ≥ 50%: recall {rec[ok[0]]:.2f} at threshold {thr[ok[0]]:.3f}")
```

OUTPUT

```
top  50 alerts: threshold 0.335 | frauds caught  23 of 83 (recall 0.28) | precision 0.46
top 100 alerts: threshold 0.153 | frauds caught  36 of 83 (recall 0.43) | precision 0.36
top 200 alerts: threshold 0.057 | frauds caught  51 of 83 (recall 0.61) | precision 0.26
top 400 alerts: threshold 0.019 | frauds caught  62 of 83 (recall 0.75) | precision 0.15
highest recall with precision ≥ 50%: recall 0.22 at threshold 0.394
```




Precision-recall curves: the plain and weighted logistic models rank almost identically (weights shift probabilities, not the ranking); LightGBM is slightly better at most recall levels.

### Expected cost

Python

```
fraud_loss, review_cost = 25000, 150                          # average fraud amount lost vs
cost of reviewing one alert
ts = np.linspace(0.002, 0.6, 300)
cost = [((best_p < t) & (yte == 1)).sum() * fraud_loss + (best_p ⩾ t).sum() * review_cost
for t in ts]
i = int(np.argmin(cost)); none = yte.sum() * fraud_loss
print(f"cost-minimising threshold {ts[i]:.3f} (theory for calibrated probabilities: review
cost ÷ fraud loss = {review_cost / fraud_loss:.3f}) | alerts {(best_p ⩾ ts[i]).sum()} |
total cost ₹{cost[i] / 1e5:.1f} lakh vs ₹{none / 1e5:.1f} lakh with no model ({1 - cost[i]
/ none:.0%} saved)")
```

OUTPUT

```
cost-minimising threshold 0.002 (theory for calibrated probabilities: review cost ÷
fraud loss = 0.006) | alerts 1480 | total cost ₹3.2 lakh vs ₹20.8 lakh with no model
(84% saved)
```

When reviewing is cheap compared with a missed fraud, the best threshold is very low: flag everything with even a small chance of fraud, as long as the team has capacity. The capacity constraint above then decides.

## When labels are scarce: anomaly detection

If confirmed fraud labels are few or arrive late, unsupervised **anomaly detection** flags unusual transactions.

Python

```
from sklearn.ensemble import IsolationForest
iso = IsolationForest(n_estimators=300, contamination=0.02, random_state=0).fit(Xtr)
anom = -iso.score_samples(Xte)
print(f"Isolation Forest (no labels used): ROC-AUC {roc_auc_score(yte, anom):.3f} | PR-AUC
{average_precision_score(yte, anom):.3f}")
```

OUTPUT

```
Isolation Forest (no labels used): ROC-AUC 0.898 | PR-AUC 0.124
```

It is weaker than a supervised model but useful for cold starts, new fraud patterns, and as an extra feature. In production, fraud systems combine rules, supervised models, anomaly scores and graph features (shared devices, cards, addresses).

## Checklist for imbalanced problems

1. Stratify splits; consider time-based validation (fraud patterns drift).
2. Report PR-AUC, recall at the operating precision or capacity, and expected cost.
3. Start with class weights or `scale_pos_weight`; try resampling inside the pipeline.
4. Choose the threshold from business constraints.
5. Recalibrate probabilities if you changed weights or resampled.
6. Monitor: base rates and fraud tactics change.

## Summary

**KEY TAKEAWAYS**

- With rare positives, accuracy is meaningless; use PR-AUC, recall/precision at the operating point and cost.
- Class weights and `scale_pos_weight` make the loss care about the rare class; they shift probabilities more than rankings.
- Undersampling, oversampling and SMOTE must happen inside the training folds (imblearn pipelines).
- Threshold moving to fit capacity, precision targets or cost is usually the most effective lever.
- Isolation Forest and other anomaly detectors help when labels are scarce.

### Quick quiz

6 questions · tap an answer to check it

**1. Fraud is 1.4% of transactions. Which metric is most informative?**

A Accuracy

B PR-AUC / recall at a fixed precision

C $R^2$

D MAE

**Answer: B.** Focus on the rare positive class.

**2. What does `class_weight="balanced"` do?**

A Weights errors on each class inversely to its frequency

B Duplicates the minority class

C Deletes majority rows

D Changes the features

**Answer: A.** The rare class counts more in the loss.

**3. When should SMOTE be applied?**

A Only to the training data within each CV fold (inside an imblearn pipeline)

B To the full dataset before splitting

C To the test set

D After deployment

**Answer: A.** Otherwise you leak test information.

**4.** The fraud team can review 100 alerts per day. How do you set the threshold?

A Take the score of the 100th-highest transaction as the cut-off

B Use 0.5

C Use the fraud rate

D Maximise accuracy

**Answer: A.** Capacity-based thresholding.

**5.** Class weighting changes recall a lot but ROC-AUC hardly at all. Why?

A Weighting mostly shifts predicted probabilities rather than the ranking of cases

B ROC-AUC is broken

C Weights remove features

D Recall is not a real metric

**Answer: A.** Similar to moving the threshold.

**6.** When is Isolation Forest useful?

A When few or no fraud labels are available

B When labels are perfect

C For regression

D For text only

**Answer: A.** It scores how unusual a record is.

EXERCISE

## Practice

(1) Use `BalancedRandomForestClassifier` from imbalanced-learn and compare PR-AUC. (2) Calibrate the weighted logistic model with `CalibratedClassifierCV` and compare Brier scores. (3) Make a time-based split by adding a transaction date and check whether PR-AUC changes.

**Show solution (worked answer)**

Python

```
from imblearn.ensemble import BalancedRandomForestClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import brier_score_loss
brf = BalancedRandomForestClassifier(300, random_state=0, sampling_strategy="all",
replacement=True, bootstrap=False).fit(Xtr, ytr)
print(average_precision_score(yte, brf.predict_proba(Xte)[:, 1]))
cal = CalibratedClassifierCV(make_pipeline(StandardScaler(),
LogisticRegression(max_iter=2000, class_weight="balanced")), cv=5).fit(Xtr, ytr)
print(brier_score_loss(yte, p_w), brier_score_loss(yte, cal.predict_proba(Xte)[:, 1]))
```

INTERVIEW CORNER

## Questions you may be asked

**Q: How do you handle imbalanced classes?** Use suitable metrics (PR-AUC, recall at precision), stratified validation, class weights or resampling inside the pipeline, strong models (boosting), and a business-driven threshold; collect more positive examples if possible.

**Q: Is SMOTE always a good idea?** No. It can create unrealistic samples, it does not help tree ensembles much, and it distorts probabilities; class weights plus threshold tuning are often enough.

**Q: Your fraud model has 99% accuracy. Is it good?** Not necessarily: with 1% fraud, predicting "no fraud" always gives 99%. Check recall, precision, PR-AUC and the cost at the chosen threshold.

## Practise more online

Free, trusted places to test yourself on this topic. Links open the original site in a new tab.