基于 PIC 临床数据库的儿童 ICU 住院死亡风险预测与分析

作者：熊美萍 **专业：**药物分析学 **学号：**2511110138

北京大学药学院

个人主页：https://meipingxiong.github.io/

2026 年 1 月 13 日

摘要

本报告基于 PIC（Paediatric Intensive Care）临床公开数据集，围绕 ICU 患者结局（二分类）预测任务开展数据分析与建模。首先对原始数据进行清洗去重，针对高缺失特征与信息过于稀疏的样本设置阈值筛除，并对数值与类别特征分别进行插补、标准化与 One-Hot 编码；同时采用按 `SUBJECT_ID` 分组的训练/测试集划分以避免数据泄漏。随后构建并比较逻辑回归、随机森林、梯度提升树及神经网络等多种模型，使用 Accuracy、Balanced Accuracy、Precision、Recall、F1、ROC-AUC 与 PR-AUC 等指标进行评估，并通过混淆矩阵、ROC 和 PR 曲线与校准曲线对模型行为进行可视化分析。实验结果表明，在类别不平衡的临床预测场景中需综合多指标与可视化诊断模型性能，才能更全面评估模型的实际效用与可靠性。

目录

1 PIC 数据集

PIC（Paediatric Intensive Care）[1] 是一个面向儿科重症医学研究的公开临床数据库，收录了重症监护相关科室患儿的结构化电子病历信息，可支持围绕预后风险评估与临床结局预测的统计分析与建模研究。课程提供的 `icu_first24hours.csv` 为基于 PIC 数据构建的特征汇总表：以一次住院记录（`HADM_ID`）为单位，将入 ICU 后前 24 小时内的高频监测与实验室检验观测进行统计聚合，从而形成可直接用于机器学习的结构化特征表示。该数据表共包含 13,258 条住院记录，覆盖 12,690 名患者；每条记录包含 1,662 个可用于建模的数值特征，包括基础人口学变量（如月龄、性别、体重）以及两类 24 小时窗口统计特征：生命体征/监测类特征以 `chart_{ID}_{stat}` 表示（共 18 个指标 ID，对应 `max/min/range` 三种统计量），实验室检验类特征以 `lab_{ID}_{stat}` 表示（共 535 个指标 ID，对应 `max/min/range` 三种统计量）。此外，数据提供院内死亡标签 `HOSPITAL_EXPIRE_FLAG`（死亡率为 5.88%），用于后续风险预测模型的训练与评估。

1.1 数据集读取

本项目使用课程提供的 PIC（Paediatric Intensive Care）特征表 `icu_first24hours.csv` 作为建模与统计分析的主要数据来源。该文件以逗号分隔格式存储，可通过 `pandas.read_csv` 直接加载为数据框。读取结果显示数据规模为 13258×1667，即包含 13258 条样本记录与 1667 个特征变量（列）。

源代码：

```
# Load the PIC first-24-hours feature table
csv_path ="./data/icu_first24hours.csv"
df =pd.read_csv(csv_path)

print("Loaded:", csv_path)
print("Shape:", df.shape) # (n_samples, n_features)
print("Columns:", len(df.columns)) # number of columns/features

# Preview the first 3 rows (wide table, may be truncated in console)
print(df.head(3))
```

输出：

```
Loaded: ./data/icu_first24hours.csv
Shape: (13258, 1667)
Columns: 1667
```

```
   SUBJECT_ID HADM_ID ADMITTIME age_month gender_is_male \
0 26 100000 2098-11-09 18:30:55 4 0
1 28 100001 2104-09-03 10:36:47 86 0
2 29 100002 2062-11-29 20:52:52 34 0

   weight_kg HOSPITAL_EXPIRE_FLAG is_early_death chart_1001_max \
0 NaN 0 0 NaN
1 NaN 1 0 NaN
2 NaN 0 0 NaN

   chart_1002_max ... lab_7017_range lab_7018_range lab_7021_range \
0 NaN ... NaN NaN NaN
1 NaN ... NaN NaN NaN
2 NaN ... NaN NaN NaN

   lab_7022_range lab_7023_range lab_7024_range lab_7025_range \
0 NaN NaN NaN NaN
1 NaN NaN NaN NaN
2 NaN NaN NaN NaN

   lab_7026_range lab_7027_range lab_7030_range
0 NaN NaN NaN
1 NaN NaN NaN
2 NaN NaN NaN

[3 rows x 1667 columns]
```

从字段结构来看，数据同时包含（1）样本索引与住院标识信息，例如 SUBJECT_ID、HADM_ID 以及入院时间戳 ADMITTIME；（2）人口学与基础信息，例如月龄 age_month、性别指示变量 gender_is_male、体重 weight_kg；（3）与临床结局相关的标注变量，例如院内死亡指示 HOSPITAL_EXPIRE_FLAG 以及早期死亡标记 is_early_death；以及（4）大量由前 24 小时监测与检验结果汇总得到的结构化特征，例如以 chart_{itemid}_max 形式表示的床旁监测统计量，以及以 lab_{itemid}_range 等形式表示的检验指标统计量。由于不同病人在 ICU 的检验与监测频率存在差异，部分变量在样本间呈现明显缺失（例如示例行中若干 chart_* 与 lab_* 特征为 NaN），后续将在数据预处理部分对缺失值模式与特征覆盖度进行系统分析与处理。

1.2 数据预处理

对于数据集中，会发现有很多的 NaN，即标签缺失情况，首先对数据集的标签缺失情况进行统计。

源代码：

```
# 特征层面的基础统计（标签缺失等）
summary =pd.DataFrame({
    "dtype": df.dtypes.astype(str),
    "non_null": df.notna().sum(),
    "missing": df.isna().sum(),
})
summary["missing_rate"] =summary["missing"] /len(df)
summary["nunique"] =df.nunique(dropna=True)

print("\n=== Feature summary (top 20 by missing_rate) ===")
print(summary.sort_values("missing_rate", ascending=False).head(20))
```

输出：

```
=== Feature summary (top 20 by missing_rate) ===
                dtype non_null missing missing_rate nunique
lab_6768_max float64 0 13258 1.000000 0
lab_6623_max float64 0 13258 1.000000 0
lab_6768_range float64 0 13258 1.000000 0
lab_6768_min float64 0 13258 1.000000 0
lab_6623_min float64 0 13258 1.000000 0
lab_6623_range float64 0 13258 1.000000 0
lab_6839_max float64 1 13257 0.999925 1
lab_6997_range float64 1 13257 0.999925 1
lab_6996_range float64 1 13257 0.999925 1
lab_6998_range float64 1 13257 0.999925 1
lab_6955_range float64 1 13257 0.999925 1
lab_6995_range float64 1 13257 0.999925 1
lab_6957_range float64 1 13257 0.999925 1
lab_6555_range float64 1 13257 0.999925 1
lab_6562_range float64 1 13257 0.999925 1
lab_6563_range float64 1 13257 0.999925 1
lab_6670_range float64 1 13257 0.999925 1
lab_6780_max float64 1 13257 0.999925 1
lab_6555_min float64 1 13257 0.999925 1
lab_6528_min float64 1 13257 0.999925 1
```

从这里我们发现，有大量的标签是缺失的，预处理阶段需要首先进行**数据清洗**。具体而言，我们的数据清洗包括**缺失率筛特征（列级）**，删除缺失率特别高的列（比如 >9% 或 >98%），这些特征覆盖太低，直接插补意义不大。此外，也要**处理样本过稀疏（行级）**，删除非缺失特征数太少的样本（比如低于某个分位数阈值，例如 <10% 的列数），否则模型噪声很大。

源代码：

```
# (1) 选择标签列（目标变量）并清理无标签样本
target =None
for t in TARGET_CANDIDATES:
    if t in df.columns:
        target =t
        break
if target is None:
    raise ValueError(f"没有找到标签列。尝试过：{TARGET_CANDIDATES}")

# 将标签转为数值；无法转的会变成 NaN
df[target] =pd.to_numeric(df[target], errors="coerce")

# 删除标签缺失的样本（因为无法用于监督学习）
df =df.dropna(subset=[target])

# 统一为 int（假设是 0/1 标签）
df[target] =df[target].astype(int)

# (2) 删除缺失率极高的特征列（列级缺失处理）
# 保护列：标签列和分组列不参与缺失率删除
protected_cols ={target, GROUP_COL}

# 统计每一列缺失率
missing_rate =df.isna().mean()

# 选出缺失率超过阈值的列（但不包括 protected_cols）
drop_cols =[
    c for c in df.columns
    if (c not in protected_cols) and (missing_rate[c] >COL_MISSING_RATE_THRESHOLD)
]

# 删除这些覆盖度极低的列
df =df.drop(columns=drop_cols)
print(f"Dropped {len(drop_cols)} columns with missing_rate > {COL_MISSING_RATE_THRESHOLD:.2f}")
print("Shape after dropping columns:", df.shape)

# (3) 删除过稀疏的样本行（行级缺失处理）
# 只在特征列上统计“非缺失比例”，不包括标签/分组列
feature_cols =[c for c in df.columns if c not in protected_cols]

# 每行非缺失特征数量
row_nonmissing =df[feature_cols].notna().sum(axis=1)

# 每行非缺失比例（非缺失特征数 / 特征总数）
```

```
row_nonmissing_ratio =row_nonmissing /max(len(feature_cols), 1)

# 过稀疏样本：非缺失比例低于阈值
row_drop_mask =row_nonmissing_ratio <ROW_NONMISSING_RATIO_THRESHOLD
print(f"Dropping {row_drop_mask.sum()} rows with non-missing ratio < {ROW_NONMISSING_RATIO_THRESHOLD:.2f}")

# 删除这些样本
df =df.loc[~row_drop_mask].reset_index(drop=True)
print("Shape after dropping sparse rows:", df.shape)
```

输出：

```
Dropped 972 columns with missing_rate > 0.98
Shape after dropping columns: (13258, 695)
Dropping 1453 rows with non-missing ratio < 0.10
Shape after dropping sparse rows: (11805, 695)
```

从输出结果看，总共舍弃了 972 列特征，保留了 695 个特征。此外舍弃了 1453 行样本，最终初步筛选后，总共有 11805 个样本，每个样本保留了 695 维的特征。接下来，我们先将数据划分为特征矩阵 X 与标签 y，并以 SUBJECT_ID 作为分组标识进行训练/测试集划分。具体而言，采用 GroupShuffleSplit 在病人级别随机抽取约 20% 的病人作为测试集，其余作为训练集；同一病人的全部样本记录仅会出现在训练集或测试集之一，从而避免患者级信息泄漏。划分后进一步比较训练集与测试集的标签正例率，以检查两者分布的一致性。

源代码：

```
# 4) 构造 X/y，并按 SUBJECT_ID 分组划分训练/测试集（防泄漏）
y =df[target].values

# 分组信息：用于 GroupShuffleSplit；若没有 SUBJECT_ID 就用行号代替
groups =df[GROUP_COL].values if GROUP_COL in df.columns else np.arange(len(df))

# 特征矩阵（去掉标签列）
X =df.drop(columns=[target])

# 按病人分组划分：同一 SUBJECT_ID 不会同时出现在 train/test
gss =GroupShuffleSplit(n_splits=1, test_size=TEST_SIZE, random_state=RANDOM_STATE)
train_idx, test_idx =next(gss.split(X, y, groups=groups))

X_train_raw, X_test_raw =X.iloc[train_idx].copy(), X.iloc[test_idx].copy()
y_train, y_test =y[train_idx], y[test_idx]

print("\nSplit summary:")
print("Train size:", X_train_raw.shape, " Test size:", X_test_raw.shape)
print("Train positive rate:", y_train.mean(), " Test positive rate:", y_test.mean())
```

输出：

```
Split summary:
Train size: (9438, 695) Test size: (2367, 695)
Train positive rate: 0.05350709896164441 Test positive rate: 0.04647232784114914
```

从划分结果可以看出，按 SUBJECT_ID 分组后训练集包含 9438 条样本、测试集包含 2367 条样本，且两者均保留 695 维原始特征（在预处理前）。该划分方式保证同一患者不会同时出现在训练集与测试集中，从而避免患者级信息泄漏。进一步比较标签分布，训练集的正例率为 5.35%，测试集的正例率为 4.65%。两者差异约为 0.71 个百分点，整体处于同一量级，表明分组随机划分后训练集与测试集在标签比例上较为接近，同时也反映出该任务存在一定的类别不平衡（正例比例约为 5%），因此后续建模中需要考虑类别不平衡处理（例如使用类别权重或阈值调节等）。

由于原始 ICU 特征表包含不同数据类型的变量（连续数值型指标与可能存在的类别/布尔型变量），且存在大量缺失值，模型训练前需要将原始表格转换为无缺失、可计算的数值特征矩阵。为保证预处理流程可复现且避免数据泄漏，本项目使用 scikit-learn 的 Pipeline 与 ColumnTransformer 构建统一的特征处理流水线，其核心目标包括：**(i) 按特征类型采用差异化处理策略；(ii) 对缺失值进行插补并显式建模缺失模式；(iii) 将类别特征编码为数值向量并对数值特征进行尺度归一化，生成可直接输入模型的矩阵表示**。具体而言，首先根据数据类型自动划分数值列与类别列：将 X_train_raw 中可解析为数值类型的列归入数值特征集合，其余列视为类别特征。随后，为减少过拟合风险并避免潜在信息泄漏，移除纯标识符列（如 SUBJECT_ID 与 HADM_ID）而不将其作为模型输入。对于数值特征，本项目采用**中位数插补**填充缺失值（SimpleImputer(strategy="median")），并启用 add_indicator=True 额外引入缺失指示变量，以显式刻画“是否缺失”的模式信息；随后使用 StandardScaler 对数值特征进行标准化以提升下游模型（例如线性模型或神经网络）的优化稳定性。对于类别特征，使用**众数插补**填充缺失，并通过 OneHotEncoder 将离散取值展开为 0/1 向量表示，其中 handle_unknown="ignore" 用于保证测试集中出现训练未见类别时不会报错。最后，使用 ColumnTransformer 将上述两套处理流程分别应用于对应列集合，并仅在训练集上执行 fit（学习插补统计量、标准化参数及类别词表），再以相同参数对测试集执行 transform，从而避免将测试集信息泄漏到预处理过程。需要注意的是，由于缺失指示变量与 One-Hot 编码会引入额外维度，预处理后的特征维度通常会相较原始列数增加。

源代码：

```
# 5) 构建预处理流水线：数值列 vs 类别列
# 自动识别数值列
numeric_cols =X_train_raw.select_dtypes(include=[np.number]).columns.tolist()

# 其余列当作类别列（包含字符串/布尔等）
categorical_cols =[c for c in X_train_raw.columns if c not in numeric_cols]

# 一般不建议把纯ID（如 SUBJECT_ID/HADM_ID）作为模型特征（容易过拟合/泄漏）
id_like =[c for c in ["SUBJECT_ID", "HADM_ID"] if c in X_train_raw.columns]
drop_from_model =id_like
```

```
# 从数值/类别列列表中移除这些ID列
numeric_cols =[c for c in numeric_cols if c not in drop_from_model]
categorical_cols =[c for c in categorical_cols if c not in drop_from_model]

# 数值特征：中位数插补 + 缺失指示变量 + 标准化
numeric_pipe =Pipeline(steps=[
    # add_indicator=True 会额外生成“是否缺失”的0/1指示特征（对医疗数据常有帮助）
    ("imputer", SimpleImputer(strategy="median", add_indicator=True)),
    ("scaler", StandardScaler())
])

# 类别特征：众数插补 + One-Hot 编码
categorical_pipe =Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore"))
])

# 将数值/类别两套处理组合起来
preprocess =ColumnTransformer(
    transformers=[
        ("num", numeric_pipe, numeric_cols),
        ("cat", categorical_pipe, categorical_cols),
    ],
    remainder="drop" # 其余列（比如ID列）直接丢弃
)

# 在训练集上拟合预处理器，然后转换 train/test
X_train =preprocess.fit_transform(X_train_raw)
X_test =preprocess.transform(X_test_raw)

print("\nAfter preprocessing:")
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
```

输出：

```
After preprocessing:
X_train shape: (9438, 1381)
X_test shape: (2367, 1381)
```

由输出结果可知，经过统一的预处理流水线后，训练集与测试集分别被转换为维度一致的特征矩阵：训练集为 9438×1381，测试集为 2367×1381。相较于预处理前的 695 维原始特征，维度增长至 1381 主要来源于特征展开操作，尤其是对存在缺失的数值特征引入的缺失指示变量（以及可能的类别特征 One-Hot 编码）。该结果表明缺失值已被插补并编码为可计算的数值表示，且训练/测试集在相同特征空间下对齐，从而能够直接用于后续预测模型的训练与评估。

1.3 统计分析

在完成数据读取、缺失处理与训练/测试集划分后，本节对数据进行统计分析与可视化，以刻画数据的基本分布特性并验证数据划分的合理性。具体而言，我们从以下几个方面展开：首先，统计训练集与测试集的标签分布并计算正例率，用于评估类别不平衡程度以及两部分数据在标签比例上的一致性；其次，分别可视化关键人口学变量（如 `age_month` 与 `gender_is_male`）在训练/测试集中的分布，以检查是否存在明显的人群结构差异；随后，从缺失数据角度分析样本级信息完整度（每条样本的非缺失特征数量/比例）与特征级覆盖度（每个特征的非缺失样本数），以揭示 ICU 数据的稀疏性并为后续特征筛选与插补策略提供依据；最后，进一步展示缺失率最高的一组特征，定位主要缺失来源并支持预处理决策的可解释性。上述可视化结果将为后续预测模型的建立与评估提供数据层面的依据。

训练/测试集标签分布：用来检查数据划分后两部分的标签比例是否一致，避免训练/测试标签分布差异导致评估偏差；同时量化类别不平衡程度（如正例约 5%），为后续采用 class weight、阈值调整等策略提供依据。从图 1 中我们可以发现，训练集与测试集中的标签分布均匀，说明了能够直接用于后续预测模型的训练与评估。

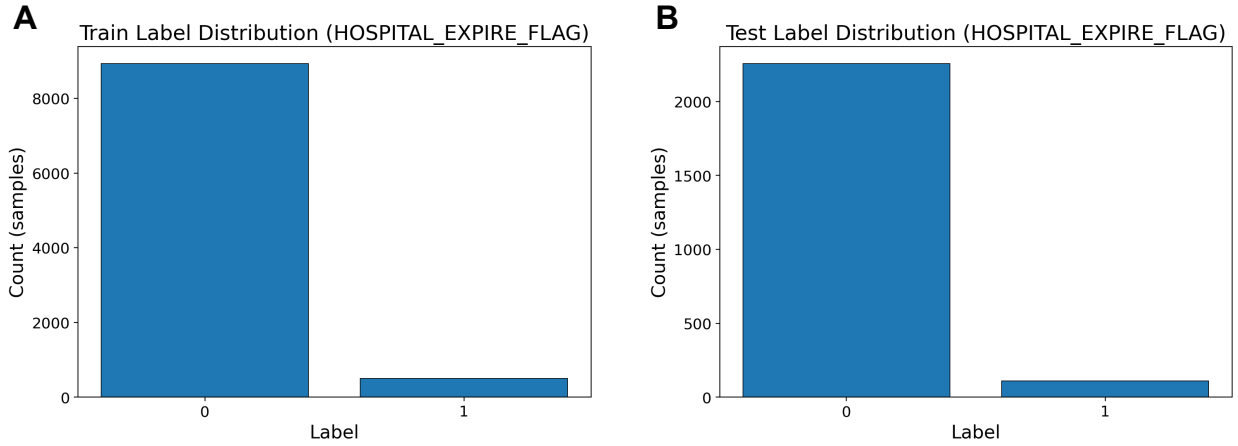

图 1. 训练集与测试集标签分布。

年龄分布：用来验证训练集与测试集在人群年龄结构上是否一致，排查因年龄段差异造成的分布偏移；同时帮助理解样本覆盖的儿科年龄范围，为后续分层分析（如不同年龄段风险差异）提供基础。从图 2 中我们可以看出，训练集与测试集的年龄分布情况相似，均是随着月份的增大，样本数量剧烈下降。

性别分布：用来检查训练/测试集性别比例是否接近，避免性别结构不一致引入分布偏移；同时可以初步观察数据中是否存在显著性别偏置，为公平性/偏差讨论提供证据。从图 3 中我们可以看出，训练集与测试集的性别分布是相似的，证明了构建的数据集没有性别结构不一致引入分布偏移问题。

小结：综合上述可视化结果可以看出，训练集与测试集在标签比例、年龄结构与性别结构上整体一致，未观察到明显的分布偏移，说明基于 `SUBJECT_ID` 的分组划分是合理的。同时，数据在样本与特征层面均存在一定缺失与稀疏性，但已在预处理阶段通过缺失列筛除、缺失值插补与缺失指示变量等策略进行了统一处理，使训练集与测试集被映射到相同的特征空间。综上，当前预处理后的数据集具备可用性与一致性，可直接用于后续预测模型的训练与评估。

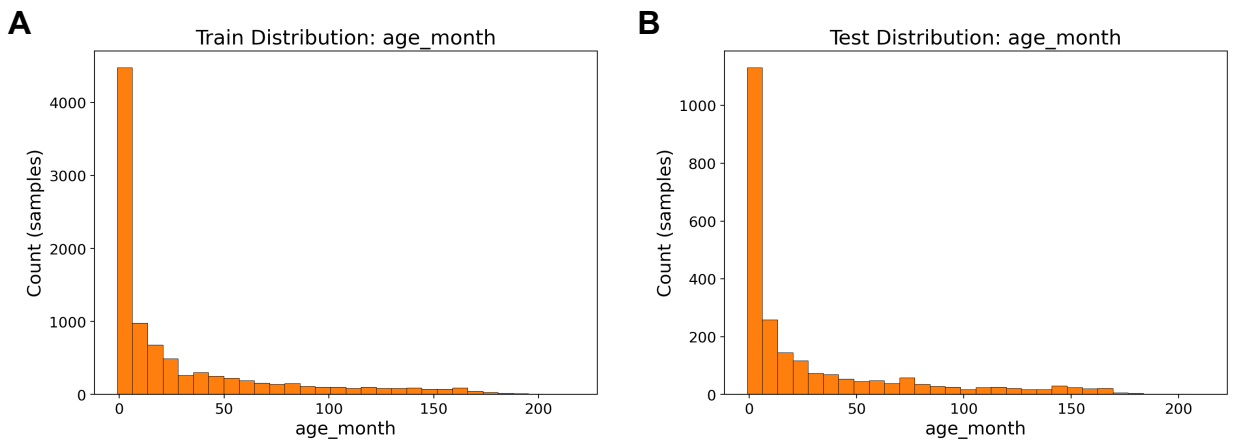

图 2. 训练集与测试集年龄分布。

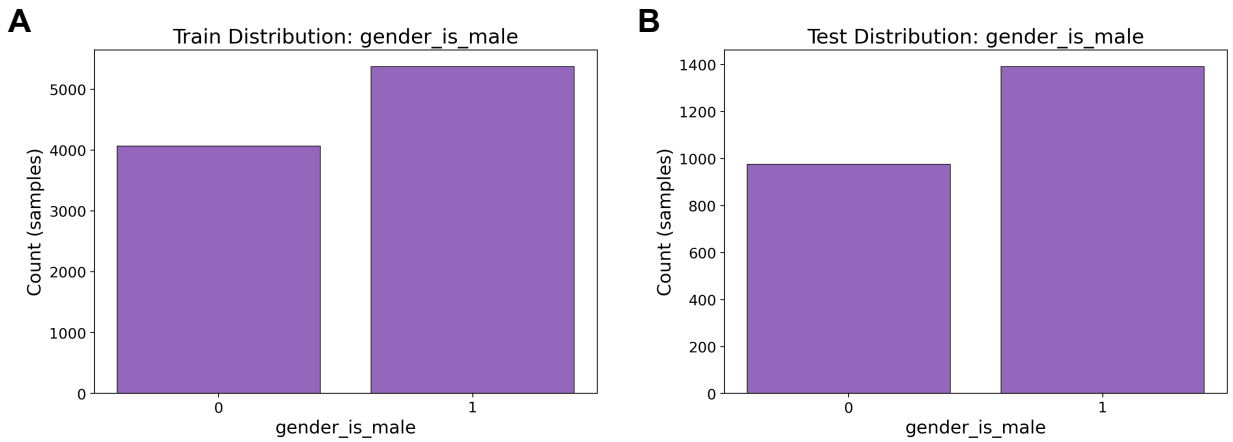

图 3. 训练集与测试集性别分布。

2 构建预测模型

2.1 预测模型建立

我们尝试了 4 种常见的机器学习算法。

2.1.1 逻辑回归（Logistic Regression, LR）

逻辑回归 [2] 是临床二分类任务中最常用的强基线模型之一。它假设输入特征的线性组合可以通过 Sigmoid 映射为“死亡风险/事件发生概率”，训练目标等价于最小化对数损失（log-loss）。在本任务中，经过缺失插补、标准化与 one-hot 编码后，LR 能在高维稀疏特征空间中稳定训练，且系数具有可解释性（可用于分析关键特征方向）。同时，LR 支持 `class_weight`，便于处理 ICU 结局预测常见的类别不平衡问题。

源代码：

```
from sklearn.linear_model import LogisticRegression
```

```
lr =LogisticRegression(
    max_iter=5000,
    class_weight=class_weight # 你前面算好的 balanced weights
)
lr.fit(X_train, y_train)

y_pred =lr.predict(X_test)
y_prob =lr.predict_proba(X_test)[:, 1]
```

2.1.2 随机森林（Random Forest, RF）

随机森林 [3] 是一类基于 Bagging 的集成树模型，通过对样本进行自助采样（bootstrap）并在每个节点随机选择特征子集进行划分，训练出多棵决策树并对结果投票/平均。它能够捕获非线性关系与特征交互，对异常值和部分缺失模式也相对鲁棒。对于 ICU 表格数据，RF 常作为“非线性强基线”。其不足在于：在高维 one-hot 特征下可能较耗时、模型体积较大，且概率输出的校准性未必理想。

源代码：

```
from sklearn.ensemble import RandomForestClassifier
import scipy.sparse as sp

# RF 通常更适合 dense；若 X_train 是稀疏矩阵则转 dense
X_train_dense =X_train.toarray() if sp.issparse(X_train) else X_train
X_test_dense =X_test.toarray() if sp.issparse(X_test) else X_test

rf =RandomForestClassifier(
    n_estimators=400,
    random_state=RANDOM_STATE,
    class_weight=class_weight,
    n_jobs=-1
)
rf.fit(X_train_dense, y_train)

y_pred =rf.predict(X_test_dense)
y_prob =rf.predict_proba(X_test_dense)[:, 1]
```

2.1.3 直方图梯度提升树（HistGradientBoosting, HGB）

HistGradientBoosting [4] 属于梯度提升决策树（GBDT）家族的高效实现，它将连续特征分箱（histogram binning）后进行分裂搜索，从而显著提升在大样本/高维特征下的训练速度。相比随机森林，GBDT 以逐步拟合残差的方式串行提升，通常具有更强的预测性能与更灵活的非线性建模能力。医疗表格数据常呈现复杂交互与非线性，HGB 能有效建模这类结构。其不足在于：对超参数较敏感，并且 `class_weight` 需要通过 `sample_weight` 方式间接实现。

源代码：

```
from sklearn.ensemble import HistGradientBoostingClassifier
import numpy as np
import scipy.sparse as sp

X_train_dense =X_train.toarray() if sp.issparse(X_train) else X_train
X_test_dense =X_test.toarray() if sp.issparse(X_test) else X_test

hgb =HistGradientBoostingClassifier(
    random_state=RANDOM_STATE,
    max_depth=6,
    learning_rate=0.05,
    max_iter=300
)

# 用 sample_weight 模拟 class_weight
sample_weight =np.where(y_train ==1, class_weight[1], class_weight[0])

hgb.fit(X_train_dense, y_train, sample_weight=sample_weight)

y_pred =hgb.predict(X_test_dense)
# HGB 有 predict_proba
y_prob =hgb.predict_proba(X_test_dense)[:, 1]
```

2.1.4 多层感知机（MLP）

MLP [5] 是基于前馈神经网络的经典表格学习模型，能够通过非线性激活函数拟合复杂决策边界。在完成插补、标准化与 one-hot 后，MLP 可作为“神经网络基线”用于对比树模型与线性模型。其优点是结构简单、实现方便；不足在于对学习率、网络深度、正则化较敏感，且对类别不平衡的原生支持有限（通常需借助 sample_weight 或重采样策略）。

源代码：

```
from sklearn.neural_network import MLPClassifier
import numpy as np
import scipy.sparse as sp

X_train_dense =X_train.toarray() if sp.issparse(X_train) else X_train
X_test_dense =X_test.toarray() if sp.issparse(X_test) else X_test

mlp =MLPClassifier(
    hidden_layer_sizes=(256, 128),
    activation="relu",
    solver="adam",
    alpha=1e-4,
    batch_size=256,
```

```
    learning_rate_init=1e-3,
    max_iter=60,
    random_state=RANDOM_STATE,
    early_stopping=True,
    n_iter_no_change=8
)

# 用 sample_weight 近似处理不平衡
sample_weight =np.where(y_train ==1, class_weight[1], class_weight[0])

mlp.fit(X_train_dense, y_train, sample_weight=sample_weight)

y_pred =mlp.predict(X_test_dense)
y_prob =mlp.predict_proba(X_test_dense)[:, 1]
```

2.1.5 表格 MLP（Tabular Neural Network）

除传统机器学习外，本实验进一步引入基于 PyTorch 的表格神经网络作为深度学习方法对照。该模型采用多层全连接网络（MLP）对预处理后的特征向量进行表征学习，并输出二分类 logit。训练使用 BCEWithLogitsLoss，并通过 pos_weight 增大少数类（死亡/事件）样本的损失权重，以缓解类别不平衡。该实现更易扩展（如加入 BatchNorm、更复杂的网络结构或更灵活的早停策略），也便于后续在深度学习框架中做进一步实验。

源代码：

```
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader
import scipy.sparse as sp
import numpy as np

device =torch.device("cuda" if torch.cuda.is_available() else "cpu")

X_train_dense =X_train.toarray() if sp.issparse(X_train) else X_train
X_test_dense =X_test.toarray() if sp.issparse(X_test) else X_test

Xtr =torch.tensor(X_train_dense, dtype=torch.float32)
Xte =torch.tensor(X_test_dense, dtype=torch.float32)
ytr =torch.tensor(y_train.reshape(-1, 1), dtype=torch.float32)
yte =torch.tensor(y_test.reshape(-1, 1), dtype=torch.float32)

train_loader =DataLoader(TensorDataset(Xtr, ytr), batch_size=256, shuffle=True)
test_loader =DataLoader(TensorDataset(Xte, yte), batch_size=512, shuffle=False)

class TabularMLP(nn.Module):
```

```
    def __init__(self, in_dim):
        super().__init__()
        self.net =nn.Sequential(
            nn.Linear(in_dim, 512),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(256, 1) # logits
        )
    def forward(self, x):
        return self.net(x)

model =TabularMLP(in_dim=X_train_dense.shape[1]).to(device)

# 不平衡：pos_weight（正类权重）
pos_weight =torch.tensor([class_weight[1] /class_weight[0]], dtype=torch.float32).to(device)
criterion =nn.BCEWithLogitsLoss(pos_weight=pos_weight)
optimizer =torch.optim.AdamW(model.parameters(), lr=1e-3, weight_decay=1e-4)

EPOCHS =20
for epoch in range(EPOCHS):
    model.train()
    for xb, yb in train_loader:
        xb, yb =xb.to(device), yb.to(device)
        optimizer.zero_grad()
        logits =model(xb)
        loss =criterion(logits, yb)
        loss.backward()
        optimizer.step()

# 推理：概率= sigmoid(logits)
model.eval()
with torch.no_grad():
    logits =model(Xte.to(device)).cpu().numpy().reshape(-1)
y_prob =1 /(1 +np.exp(-logits))
y_pred =(y_prob >=0.5).astype(int)
```

2.2 预测模型评估与可视化

本实验在完成数据清洗、缺失处理与特征工程后，分别训练了多类典型二分类模型，并在独立测试集上进行统一评估。为确保训练/测试严格隔离，数据划分采用按病人 SUBJECT_ID 分组的 GroupShuffleSplit，避免同一病人样本同时出现在训练集与测试集造成数据泄漏。评估指标涵盖分类性能与不平衡场景下的鲁棒

性，并通过多种曲线与矩阵可视化模型行为。

2.2.1 评估指标

设正类表示目标事件（如院内死亡/早期死亡），对测试集预测结果统计以下指标：

1. **Accuracy：**总体预测正确比例，反映整体分类性能；
2. **Balanced Accuracy：**对正负类召回率取平均，适用于类别不平衡情况；
3. **Precision / Recall：**分别衡量预测为正的可信度与对正类样本的覆盖能力；
4. **F1-score：**Precision 与 Recall 的调和平均，适合不平衡场景下综合比较模型；
5. **ROC-AUC：**基于不同阈值下 TPR/FPR 的面积指标，衡量排序能力；
6. **PR-AUC (Average Precision)：**在正类稀少时更敏感，反映 Precision-Recall 权衡下的整体表现。

2.2.2 预测模型评估

我们在各个指标上评估了 5 种模型，从表 1 可以看出，不同模型在各评价指标上呈现出明显差异，这也反映了 ICU 结局预测任务中类别不平衡带来的典型现象：单看 Accuracy 容易产生误判，需要结合 Balanced Accuracy、Recall、F1 以及 PR-AUC 综合判断。首先，随机森林（RF）取得了最高的 Accuracy（0.9535），同时 ROC-AUC（0.8624）和 PR-AUC（0.2237）也较高，说明其在“排序能力”（把高风险样本排到更前面）方面表现强。但其 Recall 极低（0.0091）、F1 极低（0.0178），表明在默认阈值下几乎不预测正类，属于明显“保守预测”模型：虽然总体准确率很高，但对正类（高风险/死亡）样本的识别能力几乎不可用。因此，RF 的高 Accuracy 更可能来源于“多数类占比高”，而不是模型真正具备强的临床风险识别能力。相比之下，HGB 在综合指标上更均衡：其 Accuracy（0.9286）仍然较高，同时 Precision（0.2814）与 Recall（0.3454）形成更合理的权衡，使得 F1 达到 0.3102（表中最高），并且 ROC-AUC（0.8397）与 PR-AUC（0.2161）也保持在较高水平。整体来看，HGB 更符合“不平衡二分类任务”对模型的实际需求：既具备较强区分能力，也能在一定程度上识别正类样本。在神经网络方法中，Table Neural Network 的 Balanced Accuracy（0.7049）最高，且 Recall（0.5454）也较高，说明它对正类更加敏感、更倾向于识别高风险样本；相应地，其 Precision（0.1639）不高，导致 F1（0.2521）略低于 HGB，但整体属于“更关注召回”的模型风格，适合需要尽量减少漏报的应用需求。相比之下，MLP 虽然 Accuracy（0.9518）较高，但 Balanced Accuracy（0.5380）与 F1（0.1363）较低，说明其在不平衡数据下仍然存在偏向多数类的倾向。

表 1. 不同机器学习模型评价指标

模型	Accuracy	Balanced Accuracy	Precision	Recall	F1	ROC AUC	PR AUC
逻辑回归 [2]	0.7955	0.6722	0.1199	0.5363	0.1960	0.7317	0.1406
随机森林 [3]	**0.9535**	0.5043	**0.5000**	0.0091	0.0178	**0.8624**	**0.2237**
HGB [4]	0.9286	0.6512	0.2814	0.3454	**0.3102**	0.8397	0.2161
MLP [5]	0.9518	0.5380	0.4090	0.0818	0.1363	0.7812	0.1709
Table Neural Network [5]	0.8495	**0.7049**	0.1639	**0.5454**	0.2521	0.8063	0.1954

综上，若以不平衡场景下更可靠的指标（F1、PR-AUC、Balanced Accuracy）作为主要依据，HGB 在整体权衡上最优（F1 最高且 AUC 较高），而 Table Neural Network 在正类召回与 Balanced Accuracy 上更突出，适用于更强调“少漏诊”的场景；RF 虽然 ROC/PR 表现强，但在默认阈值下几乎不预测正类，需通过阈值调整/校准后才可能转化为可用的分类器。

2.2.3 预测模型可视化分析

为了全面评估不同预测模型在 ICU 结局预测任务中的表现，仅依赖单一数值指标（如 Accuracy 或 AUC）往往难以揭示模型的实际行为和潜在风险。因此，本研究从多个互补角度对模型预测结果进行可视化分析，以刻画模型在不平衡数据场景下的判别特性、错误模式以及概率输出的可靠性。具体而言，模型可视化分析主要从以下四个方面展开：**（1）混淆矩阵（Confusion Matrix）**混淆矩阵用于直观展示模型在固定分类阈值下对正负样本的预测结果分布，包括正确分类与误分类的比例。通过按真实标签进行归一化，可以清晰观察模型在正类与负类上的召回情况，识别模型是否存在明显的漏诊（False Negative）或误报（False Positive）倾向。该分析对于医疗风险预测尤为重要，因为漏诊与误报在实际应用中具有不同的临床代价。**（2）ROC 曲线（Receiver Operating Characteristic Curve）**ROC 曲线刻画了模型在不同决策阈值下真阳性率与假阳性率之间的权衡关系，用于评估模型对正负样本的整体区分能力。ROC-AUC 反映的是模型的排序性能，与具体阈值选择无关，有助于判断模型是否具备将高风险样本排在前列的潜在能力。**（3）Precision–Recall 曲线（PR Curve）**在正负样本比例高度不平衡的 ICU 数据中，PR 曲线相比 ROC 曲线更能反映模型对少数类（高风险样本）的识别效果。通过分析 Precision 与 Recall 的变化趋势，可以评估模型在提高正类召回率的同时，是否引入过多误报，从而判断其在实际应用中的可用区间。**（4）概率校准曲线（Calibration Curve）**概率校准曲线用于评估模型输出的预测概率与真实事件发生频率之间的一致性。良好的概率校准意味着模型不仅能够区分样本风险高低，其输出的概率值也可以被解释为可信的风险水平。该分析对于需要基于预测概率进行风险分层或决策支持的医疗场景尤为关键。通过上述多角度的可视化分析，可以更全面地理解不同模型的行为特征及其优缺点，为后续模型选择、阈值调整及潜在的临床应用提供依据。

如图 4 所示，从混淆矩阵来看，RF 和 MLP 几乎把所有样本都判成负类（负类接近全对，但正类几乎全错）。这代表模型在默认阈值下极其保守，几乎没有正类识别能力（Recall 0），对“高风险预警”用途不合格。Tabular Neural Network 相对更均衡。负类仍然较强，同时正类也能抓到一部分（TP 有明显占比），整体比 LogReg/HGB 更像“正类可被识别”的模型之一；但仍能看到正类漏报存在，说明阈值/损失权重虽有改善，但还不完美。

ROC 主要看模型对正负样本的排序能力（阈值可变）。随机森林的 ROC 分数质量好，但需要调阈值或做概率校准/代价敏感决策，否则分类结果很差。HGB 的 ROC 很强且曲线平滑，说明提升树模型在该任务上具有很好的区分能力，并且比 RF 更容易通过阈值调整获得合理的召回。Tabular Neural Network 的 ROC 表现良好，略弱于 RF/HGB，但明显强于线性基线，说明深度模型在此任务上确实学到了一定的可分结构。

PR 曲线（尤其 AP）更贴近 ICU 这种正类稀缺任务，因为它强调“预测为正时到底有多可信”。PR 表现相对最好/并列最好之一，说明当 RF 给出较高分时，它的正类命中率其实不错。但默认阈值下几乎不报正，属于“只在极少数非常确定时才报正”的模型风格。HGB 与 RF 非常接近，并且 PR 曲线形态通常更平滑，意味着在不同召回区间都有一定可用性。对不平衡问题，HGB 往往是很强的通用选择。Tabular Neural

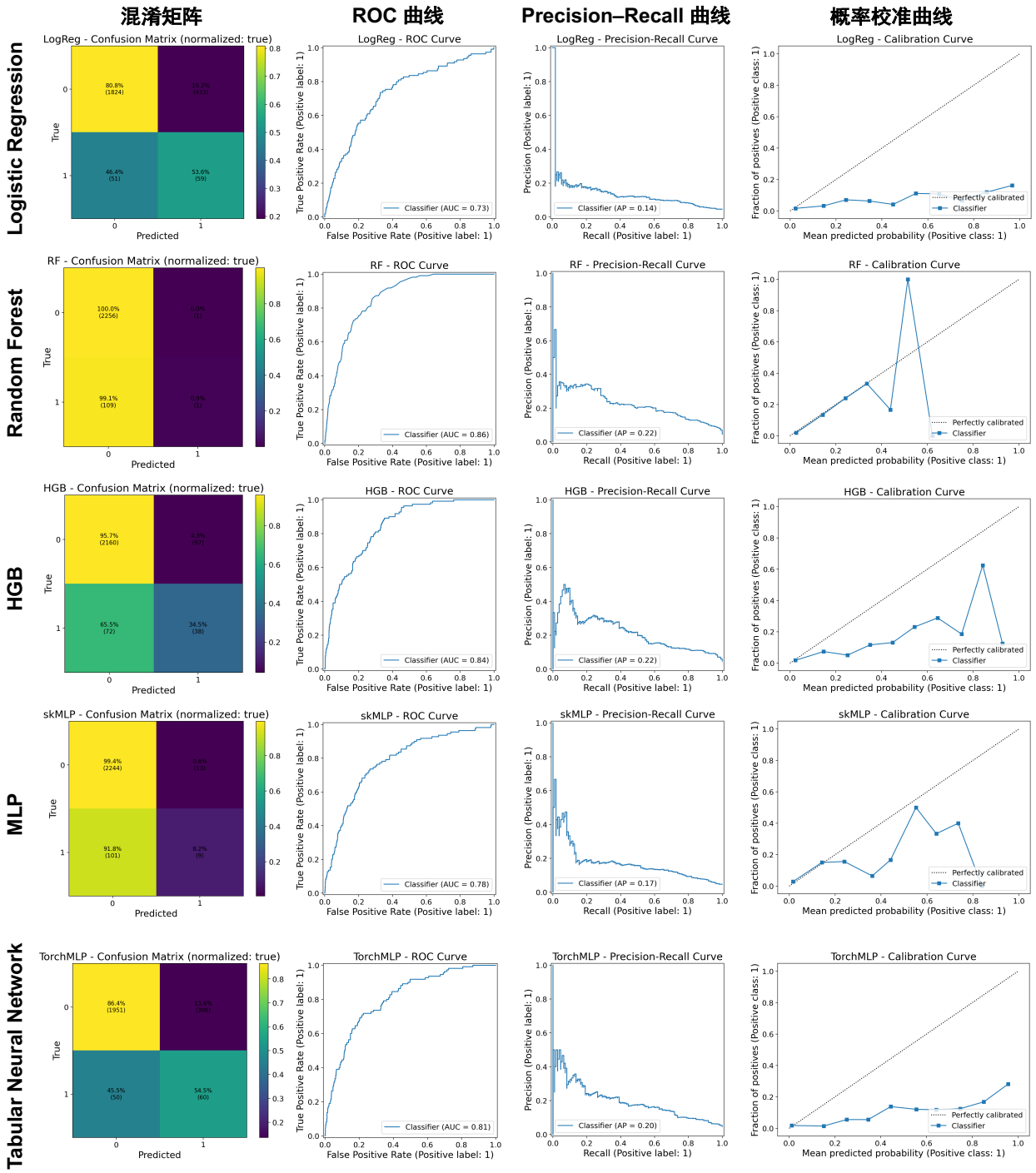

图 4. 不同预测模型在测试集上的性能表现

Network 略低于 RF/HGB，但整体仍显著优于 LogReg；说明它在提高召回时 Precision 会较快下降，但仍有一定实用区间。

校准曲线反映“预测概率”能否当作真正风险解释。图 4中各模型的校准曲线的共同特点是：多数模型在中高概率区域波动明显，说明概率校准整体一般（这在非校准树模型/不平衡训练中很常见）。

3 项目展示网页

本研究的项目展示网页请见 https://meipingxiong.github.io/PIC_anlysis。

4 结论

本报告围绕 PIC（Paediatric Intensive Care）数据集完成了从数据读取、清洗预处理到预测建模与评估的完整流程。预处理中针对临床数据缺失严重的特点，先进行列级/行级缺失筛除，并采用插补、标准化与 One-Hot 编码构建统一特征表示；同时使用按 SUBJECT_ID 分组划分训练/测试集，避免同一病人样本跨集合造成数据泄漏，保证评估结果可信。在模型层面，实验比较了逻辑回归、随机森林、梯度提升树以及神经网络等多种方法。结果表明，在类别不平衡的 ICU 结局预测任务中，仅依赖 Accuracy 容易被多数类主导而产生误导，应结合 Balanced Accuracy、F1、ROC-AUC 与 PR-AUC 综合判断模型能力。配合混淆矩阵、ROC/PR 曲线和校准曲线的可视化分析，可以进一步揭示不同模型在漏报/误报取舍、排序能力以及概率可靠性方面的差异，从而为模型选择与阈值设定提供依据。最后引入 SHAP 解释对关键特征贡献进行分析，增强了模型预测结果的可解释性与分析价值。

参考文献

[1] Xian Zeng, Gang Yu, Yang Lu, Linhua Tan, Xiujing Wu, Shanshan Shi, Huilong Duan, Qiang Shu, and Haomin Li. Pic, a paediatric-specific intensive care database. *Scientific data*, 7(1):14, 2020.

[2] David R Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 20(2):215–232, 1958.

[3] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[4] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[5] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

A 附录代码

```
import os
import numpy as np
import pandas as pd


from sklearn.model_selection import GroupShuffleSplit
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.utils.class_weight import compute_class_weight
```

```
import matplotlib.pyplot as plt

# ==========================
# 配置区（你可按报告需要调整）
# ==========================
CSV_PATH ="./data/icu_first24hours.csv"
fig_dir ="./figures"

# 标签候选列：优先用 HOSPITAL_EXPIRE_FLAG，如果没有再用 is_early_death
TARGET_CANDIDATES =["HOSPITAL_EXPIRE_FLAG", "is_early_death"]

# 按病人ID分组划分，防止同一病人进入训练集和测试集造成数据泄漏
GROUP_COL ="SUBJECT_ID"

# 缺失值阈值：缺失率超过该阈值的列直接删除（覆盖度太低）
COL_MISSING_RATE_THRESHOLD =0.98

# 稀疏样本阈值：某行（样本）非缺失特征比例低于该阈值则删除
ROW_NONMISSING_RATIO_THRESHOLD =0.10

# 测试集比例 & 随机种子
TEST_SIZE =0.2
RANDOM_STATE =42

# ==========================
# 1) 读取数据
# ==========================
if not os.path.exists(CSV_PATH):
    raise FileNotFoundError(f"找不到文件：{CSV_PATH}")

df =pd.read_csv(CSV_PATH)
print("Loaded:", CSV_PATH)
print("Shape:", df.shape)
print("Columns:", len(df.columns))

# ==========================
# 2) 基础清洗：时间字段解析 + 去重
# ==========================
# 如果存在入院时间字段，则尝试解析为 datetime（解析失败则变为 NaT）
if "ADMITTIME" in df.columns:
    df["ADMITTIME"] =pd.to_datetime(df["ADMITTIME"], errors="coerce")

# 去除完全重复的行（作为安全基线）
df =df.drop_duplicates()
print("After drop_duplicates:", df.shape)
```

```
# =========================
# 3) 选择标签列（目标变量）并清理无标签样本
# =========================
target =None
for t in TARGET_CANDIDATES:
    if t in df.columns:
        target =t
        break
if target is None:
    raise ValueError(f"没有找到标签列。尝试过：{TARGET_CANDIDATES}")

# 将标签转为数值；无法转的会变成 NaN
df[target] =pd.to_numeric(df[target], errors="coerce")

# 删除标签缺失的样本（因为无法用于监督学习）
df =df.dropna(subset=[target])

# 统一为 int（假设是 0/1 标签）
df[target] =df[target].astype(int)

# =========================
# 4) 删除缺失率极高的特征列（列级缺失处理）
# =========================
# 保护列：标签列和分组列不参与缺失率删除
protected_cols ={target, GROUP_COL}

# 统计每一列缺失率
missing_rate =df.isna().mean()

# 选出缺失率超过阈值的列（但不包括 protected_cols）
drop_cols =[
    c for c in df.columns
    if (c not in protected_cols) and (missing_rate[c] >COL_MISSING_RATE_THRESHOLD)
]

# 删除这些覆盖度极低的列
df =df.drop(columns=drop_cols)
print(f"Dropped {len(drop_cols)} columns with missing_rate > {COL_MISSING_RATE_THRESHOLD:.2f}")
print("Shape after dropping columns:", df.shape)

# =========================
# 5) 删除过稀疏的样本行（行级缺失处理）
# =========================
# 只在特征列上统计“非缺失比例”，不包括标签/分组列
```

```
feature_cols =[c for c in df.columns if c not in protected_cols]

# 每行非缺失特征数量
row_nonmissing =df[feature_cols].notna().sum(axis=1)

# 每行非缺失比例（非缺失特征数 / 特征总数）
row_nonmissing_ratio =row_nonmissing /max(len(feature_cols), 1)

# 过稀疏样本：非缺失比例低于阈值
row_drop_mask =row_nonmissing_ratio <ROW_NONMISSING_RATIO_THRESHOLD
print(f"Dropping {row_drop_mask.sum()} rows with non-missing ratio < {ROW_NONMISSING_RATIO_THRESHOLD:.2f}")

# 删除这些样本
df =df.loc[~row_drop_mask].reset_index(drop=True)
print("Shape after dropping sparse rows:", df.shape)

# ==========================
# 6) （可选）从时间字段提取简单特征，并移除原始时间戳
# ==========================
if "ADMITTIME" in df.columns:
    # 提取入院小时、星期几等低维特征
    df["admit_hour"] =df["ADMITTIME"].dt.hour
    df["admit_weekday"] =df["ADMITTIME"].dt.weekday

    # 原始时间戳通常不直接喂给模型（高基数、也不便于解释），这里删除
    df =df.drop(columns=["ADMITTIME"])

# ==========================
# 7) 构造 X/y，并按 SUBJECT_ID 分组划分训练/测试集（防泄漏）
# ==========================
y =df[target].values

# 分组信息：用于 GroupShuffleSplit；若没有 SUBJECT_ID 就用行号代替
groups =df[GROUP_COL].values if GROUP_COL in df.columns else np.arange(len(df))

# 特征矩阵（去掉标签列）
X =df.drop(columns=[target])

# 按病人分组划分：同一 SUBJECT_ID 不会同时出现在 train/test
gss =GroupShuffleSplit(n_splits=1, test_size=TEST_SIZE, random_state=RANDOM_STATE)
train_idx, test_idx =next(gss.split(X, y, groups=groups))

X_train_raw, X_test_raw =X.iloc[train_idx].copy(), X.iloc[test_idx].copy()
y_train, y_test =y[train_idx], y[test_idx]
```

```
print("\nSplit summary:")
print("Train size:", X_train_raw.shape, " Test size:", X_test_raw.shape)
print("Train positive rate:", y_train.mean(), " Test positive rate:", y_test.mean())

# =========================
# 8) 构建预处理流水线：数值列 vs 类别列
# =========================
# 自动识别数值列
numeric_cols =X_train_raw.select_dtypes(include=[np.number]).columns.tolist()

# 其余列当作类别列（包含字符串/布尔等）
categorical_cols =[c for c in X_train_raw.columns if c not in numeric_cols]

# 一般不建议把纯ID（如 SUBJECT_ID/HADM_ID）作为模型特征（容易过拟合/泄漏）
id_like =[c for c in ["SUBJECT_ID", "HADM_ID"] if c in X_train_raw.columns]
drop_from_model =id_like

# 从数值/类别列列表中移除这些ID列
numeric_cols =[c for c in numeric_cols if c not in drop_from_model]
categorical_cols =[c for c in categorical_cols if c not in drop_from_model]

# 数值特征：中位数插补 + 缺失指示变量 + 标准化
numeric_pipe =Pipeline(steps=[
    # add_indicator=True 会额外生成“是否缺失”的0/1指示特征（对医疗数据常有帮助）
    ("imputer", SimpleImputer(strategy="median", add_indicator=True)),
    ("scaler", StandardScaler())
])

# 类别特征：众数插补 + One-Hot 编码
categorical_pipe =Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore"))
])

# 将数值/类别两套处理组合起来
preprocess =ColumnTransformer(
    transformers=[
        ("num", numeric_pipe, numeric_cols),
        ("cat", categorical_pipe, categorical_cols),
    ],
    remainder="drop" # 其余列（比如ID列）直接丢弃
)

# 在训练集上拟合预处理器，然后转换 train/test
X_train =preprocess.fit_transform(X_train_raw)
```

```
X_test =preprocess.transform(X_test_raw)

print("\nAfter preprocessing:")
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)

# ==========================
# 9) （可选）类别不平衡：计算 balanced class weights
# ==========================
classes =np.array([0, 1])
cw =compute_class_weight(class_weight="balanced", classes=classes, y=y_train)
class_weight ={0: cw[0], 1: cw[1]}
print("\nClass weight (balanced):", class_weight)

# 到这里你已经得到：
# - X_train, X_test: 预处理后的特征矩阵（可直接喂给 sklearn 模型）
# - y_train, y_test: 标签
# - class_weight: 可用于处理类别不平衡（如逻辑回归、SVM 等支持 class_weight 的模型）

# ==========================
# 10) 分析与可视化（字体放大 + 同类同色）
# ==========================
os.makedirs(fig_dir, exist_ok=True)

# ---- 全局字体大小设置（你可按需要继续调大/调小）----
plt.rcParams.update({
    "figure.figsize": (8, 5.5), # 图的尺寸更大
    "axes.titlesize": 18, # 标题字号
    "axes.labelsize": 16, # 坐标轴标题字号
    "xtick.labelsize": 13, # x轴刻度字号
    "ytick.labelsize": 13, # y轴刻度字号
    "legend.fontsize": 13, # 图例字号（如果用到）
})

# ---- 同一类图用同一颜色（你也可以替换成自己喜欢的 tab:xxx）----
COLORS ={
    "label": "tab:blue", # 标签分布：train/test 同色
    "age": "tab:orange", # 年龄分布
    "gender": "tab:purple", # 性别分布
    "row_missing": "tab:green", # 样本非缺失情况
    "coverage": "tab:brown", # 特征覆盖度
    "topk": "tab:red", # Top缺失率特征条形图
}

def save_show(fig_name: str):
```

```
    """保存并显示当前图像。"""
    out_path =os.path.join("eval_figures", fig_name)
    plt.savefig(out_path, dpi=200, bbox_inches="tight")
    plt.show()
    print("Saved:", out_path)

# =========================
# 10.1 标签分布：Train/Test（同类同色）
# =========================
train_counts =pd.Series(y_train).value_counts().sort_index()
test_counts =pd.Series(y_test).value_counts().sort_index()

plt.figure()
plt.bar(train_counts.index.astype(str), train_counts.values,
        color=COLORS["label"], edgecolor="black", linewidth=0.6)
plt.title(f"Train Label Distribution ({target})")
plt.xlabel("Label")
plt.ylabel("Count (samples)")
save_show("dist_train_label.png")

plt.figure()
plt.bar(test_counts.index.astype(str), test_counts.values,
        color=COLORS["label"], edgecolor="black", linewidth=0.6)
plt.title(f"Test Label Distribution ({target})")
plt.xlabel("Label")
plt.ylabel("Count (samples)")
save_show("dist_test_label.png")

print(f"[Label rate] Train positive rate: {np.mean(y_train):.4f}, Test positive rate: {np.mean(y_test):.4f}")

# =========================
# 10.2 年龄分布（如果有 age_month）
# =========================
if "age_month" in X_train_raw.columns:
    plt.figure()
    plt.hist(X_train_raw["age_month"].dropna().values, bins=30,
             color=COLORS["age"], edgecolor="black", linewidth=0.4)
    plt.title("Train Distribution: age_month")
    plt.xlabel("age_month")
    plt.ylabel("Count (samples)")
    save_show("dist_train_age_month.png")

    plt.figure()
    plt.hist(X_test_raw["age_month"].dropna().values, bins=30,
             color=COLORS["age"], edgecolor="black", linewidth=0.4)
```

```
    plt.title("Test Distribution: age_month")
    plt.xlabel("age_month")
    plt.ylabel("Count (samples)")
    save_show("dist_test_age_month.png")

# =========================
# 10.3 性别分布（如果有 gender_is_male）
# =========================
if "gender_is_male" in X_train_raw.columns:
    tr_gender =X_train_raw["gender_is_male"].value_counts(dropna=False).sort_index()
    te_gender =X_test_raw["gender_is_male"].value_counts(dropna=False).sort_index()

    plt.figure()
    plt.bar(tr_gender.index.astype(str), tr_gender.values,
            color=COLORS["gender"], edgecolor="black", linewidth=0.6)
    plt.title("Train Distribution: gender_is_male")
    plt.xlabel("gender_is_male")
    plt.ylabel("Count (samples)")
    save_show("dist_train_gender_is_male.png")

    plt.figure()
    plt.bar(te_gender.index.astype(str), te_gender.values,
            color=COLORS["gender"], edgecolor="black", linewidth=0.6)
    plt.title("Test Distribution: gender_is_male")
    plt.xlabel("gender_is_male")
    plt.ylabel("Count (samples)")
    save_show("dist_test_gender_is_male.png")

# =========================
# 10.4 样本信息完整度：每行非缺失数/非缺失比例（Train/Test 同类同色）
# =========================
def plot_row_missingness(X_raw: pd.DataFrame, split_name: str):
    row_nonmissing =X_raw.notna().sum(axis=1)
    row_nonmissing_ratio =row_nonmissing /X_raw.shape[1]

    plt.figure()
    plt.hist(row_nonmissing.values, bins=30,
             color=COLORS["row_missing"], edgecolor="black", linewidth=0.4)
    plt.title(f"{split_name}: # Non-missing Features per Sample")
    plt.xlabel("# Non-missing features")
    plt.ylabel("Count (samples)")
    save_show(f"missing_{split_name.lower()}_row_nonmissing_count.png")

    plt.figure()
    plt.hist(row_nonmissing_ratio.values, bins=30,
```

```
            color=COLORS["row_missing"], edgecolor="black", linewidth=0.4)
    plt.title(f"{split_name}: Non-missing Feature Ratio per Sample")
    plt.xlabel("Non-missing ratio")
    plt.ylabel("Count (samples)")
    save_show(f"missing_{split_name.lower()}_row_nonmissing_ratio.png")

plot_row_missingness(X_train_raw, "Train")
plot_row_missingness(X_test_raw, "Test")

# ==========================
# 10.5 特征覆盖度（训练集上统计更合理）
# ==========================
col_coverage =X_train_raw.notna().sum(axis=0)

plt.figure()
plt.hist(col_coverage.values, bins=30,
         color=COLORS["coverage"], edgecolor="black", linewidth=0.4)
plt.title("Train Feature Coverage: # Non-missing Samples per Feature")
plt.xlabel("# Non-missing samples (train)")
plt.ylabel("Count (features)")
save_show("missing_train_feature_coverage_hist.png")

# ==========================
# 10.6 缺失率最高 Top-K 特征（条形图）
# ==========================
train_missing_rate =X_train_raw.isna().mean().sort_values(ascending=False)
TOPK =20
topk =train_missing_rate.head(TOPK)[::-1] # 反转后从低到高画，更易读

plt.figure()
plt.barh(topk.index.astype(str), topk.values,
         color=COLORS["topk"], edgecolor="black", linewidth=0.4)
plt.title(f"Top-{TOPK} Features by Missing Rate (Train)")
plt.xlabel("Missing rate")
plt.ylabel("Feature")
save_show("missing_train_topk_features_barh.png")

print("\n[Missing summary] Train missing rate describe():")
print(train_missing_rate.describe())

# ==========================
# 11) 预测模型训练 + 统一评估与可视化
# ==========================
import scipy.sparse as sp
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    roc_auc_score, average_precision_score, balanced_accuracy_score,
    confusion_matrix, RocCurveDisplay, PrecisionRecallDisplay
)
from sklearn.calibration import CalibrationDisplay



# ---------- 工具函数：稀疏/稠密兼容 ----------
def to_dense_if_needed(X, max_dense_elems=2e8):
    """
    将稀疏矩阵转为 dense（用于 MLP / PyTorch）。
    max_dense_elems: 最大允许元素数量（避免 OOM），默认 2e8 ~ 200M floats.
    """
    if sp.issparse(X):
        n_elems =X.shape[0] *X.shape[1]
        if n_elems >max_dense_elems:
            raise MemoryError(f"矩阵过大不适合转 dense: {X.shape}, elems={n_elems}")
        return X.toarray()
    return X

def get_score_and_prob(model, X):
    """
    返回:
      - y_score: 用于 ROC/PR 的连续分数
      - y_prob: 预测为正类的概率（若可得）
    """
    y_prob =None
    if hasattr(model, "predict_proba"):
        y_prob =model.predict_proba(X)[:, 1]
        y_score =y_prob
    elif hasattr(model, "decision_function"):
        y_score =model.decision_function(X)
    else:
        # fallback：用预测标签当分数（不推荐，但保证代码不断）
        y_score =model.predict(X)
    return y_score, y_prob

def plot_confusion_matrix(cm, title, fig_name, normalize="true"):
    """
    cm: confusion_matrix 输出（计数）
```

```
    normalize:
      - "true": 按真实标签归一化（每行=100%）【推荐】
      - "pred": 按预测标签归一化（每列=100%）
      - "all" : 全局归一化（总和=100%）
      - None : 不归一化（显示计数）
    """
    cm =np.asarray(cm, dtype=float)

    if normalize =="true":
        denom =cm.sum(axis=1, keepdims=True)
        cm_norm =np.divide(cm, denom, out=np.zeros_like(cm), where=denom !=0)
    elif normalize =="pred":
        denom =cm.sum(axis=0, keepdims=True)
        cm_norm =np.divide(cm, denom, out=np.zeros_like(cm), where=denom !=0)
    elif normalize =="all":
        denom =cm.sum()
        cm_norm =cm /denom if denom !=0 else np.zeros_like(cm)
    else:
        cm_norm =cm

    plt.figure()
    plt.imshow(cm_norm, interpolation="nearest")
    plt.title(title +("" if normalize is None else f" (normalized: {normalize})"))
    plt.colorbar()

    tick_marks =np.arange(2)
    plt.xticks(tick_marks, ["0", "1"])
    plt.yticks(tick_marks, ["0", "1"])
    plt.xlabel("Predicted")
    plt.ylabel("True")

    # 单元格文字：百分比 + (原始计数)
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            if normalize is None:
                txt =f"{int(cm[i, j])}"
            else:
                txt =f"{cm_norm[i, j]*100:.1f}%\n({int(cm[i, j])})"
            plt.text(j, i, txt, ha="center", va="center")

    save_show(fig_name)

def evaluate_binary_classifier(model, X_tr, y_tr, X_te, y_te, model_name):
    """
    统一评估：输出 metrics + 画图（混淆矩阵/ROC/PR/校准）
```

```
"""
# 预测
y_pred =model.predict(X_te)
y_score, y_prob =get_score_and_prob(model, X_te)

# 指标（注意：AUC 需要 y_score 连续值）
metrics ={
    "model": model_name,
    "acc": accuracy_score(y_te, y_pred),
    "balanced_acc": balanced_accuracy_score(y_te, y_pred),
    "precision": precision_score(y_te, y_pred, zero_division=0),
    "recall": recall_score(y_te, y_pred, zero_division=0),
    "f1": f1_score(y_te, y_pred, zero_division=0),
}
# AUC / PR-AUC 可能因为只有单类等原因报错，做保护
try:
    metrics["roc_auc"] =roc_auc_score(y_te, y_score)
except Exception:
    metrics["roc_auc"] =np.nan
try:
    metrics["pr_auc"] =average_precision_score(y_te, y_score)
except Exception:
    metrics["pr_auc"] =np.nan

# --- 1) 混淆矩阵 ---
cm =confusion_matrix(y_te, y_pred, labels=[0, 1])
plot_confusion_matrix(
    cm,
    title=f"{model_name} - Confusion Matrix",
    fig_name=f"eval_{model_name}_confusion_matrix.png"
)

# --- 2) ROC 曲线 ---
try:
    RocCurveDisplay.from_predictions(y_te, y_score)
    plt.title(f"{model_name} - ROC Curve")
    save_show(f"eval_{model_name}_roc.png")
except Exception as e:
    print(f"[Warn] ROC plot failed for {model_name}: {e}")

# --- 3) PR 曲线 ---
try:
    PrecisionRecallDisplay.from_predictions(y_te, y_score)
    plt.title(f"{model_name} - Precision-Recall Curve")
    save_show(f"eval_{model_name}_pr.png")
```

```
    except Exception as e:
        print(f"[Warn] PR plot failed for {model_name}: {e}")

    # --- 4) 校准曲线（需要概率更合理；没有 predict_proba 则跳过） ---
    if y_prob is not None:
        try:
            CalibrationDisplay.from_predictions(y_te, y_prob, n_bins=10)
            plt.title(f"{model_name} - Calibration Curve")
            save_show(f"eval_{model_name}_calibration.png")
        except Exception as e:
            print(f"[Warn] Calibration plot failed for {model_name}: {e}")

    return metrics

# ==========================
# 11.1 训练若干 sklearn 典型模型
# ==========================
results =[]

# (A) 逻辑回归：强基线（可用 class_weight）
lr =LogisticRegression(max_iter=5000, class_weight=class_weight, n_jobs=None)
lr.fit(X_train, y_train)
results.append(evaluate_binary_classifier(lr, X_train, y_train, X_test, y_test, "LogReg"))

# (B) 随机森林：非线性、鲁棒（注意：RF 需要 dense）
X_train_dense =to_dense_if_needed(X_train)
X_test_dense =to_dense_if_needed(X_test)

rf =RandomForestClassifier(
    n_estimators=400,
    random_state=RANDOM_STATE,
    class_weight=class_weight,
    n_jobs=-1,
    max_depth=None
)
rf.fit(X_train_dense, y_train)
results.append(evaluate_binary_classifier(rf, X_train_dense, y_train, X_test_dense, y_test, "RF"))

# (C) HistGradientBoosting：强力树模型（需要 dense；且它用 sample_weight 而不是 class_weight）
hgb =HistGradientBoostingClassifier(
    random_state=RANDOM_STATE,
    max_depth=6,
    learning_rate=0.05,
    max_iter=300
)
```

```
# 用 sample_weight 模拟 class_weight
sample_weight =np.where(y_train ==1, class_weight[1], class_weight[0])
hgb.fit(X_train_dense, y_train, sample_weight=sample_weight)
results.append(evaluate_binary_classifier(hgb, X_train_dense, y_train, X_test_dense, y_test, "HGB"))

# (D) 可选：XGBoost（如果你装了 xgboost，就能跑；否则自动跳过）
try:
    from xgboost import XGBClassifier
    xgb =XGBClassifier(
        n_estimators=600,
        max_depth=6,
        learning_rate=0.05,
        subsample=0.9,
        colsample_bytree=0.9,
        reg_lambda=1.0,
        random_state=RANDOM_STATE,
        eval_metric="logloss",
        n_jobs=-1,
        # 用 scale_pos_weight 处理类别不平衡（近似）
        scale_pos_weight=(class_weight[1] /class_weight[0])
    )
    xgb.fit(X_train_dense, y_train)
    results.append(evaluate_binary_classifier(xgb, X_train_dense, y_train, X_test_dense, y_test, "XGB"))
except Exception as e:
    print("[Info] xgboost not available, skip XGB. Error:", e)

# (E) 可选：sklearn MLP（小型神经网络基线；需要 dense）
try:
    from sklearn.neural_network import MLPClassifier
    mlp =MLPClassifier(
        hidden_layer_sizes=(256, 128),
        activation="relu",
        solver="adam",
        alpha=1e-4,
        batch_size=256,
        learning_rate_init=1e-3,
        max_iter=60,
        random_state=RANDOM_STATE,
        early_stopping=True,
        n_iter_no_change=8
    )
    # sklearn 的 MLP 没有 class_weight；用 sample_weight 近似
    mlp.fit(X_train_dense, y_train)
    results.append(evaluate_binary_classifier(mlp, X_train_dense, y_train, X_test_dense, y_test, "skMLP"))
except Exception as e:
```

```
    print("[Info] sklearn MLP failed/skip:", e)

# =========================
# 11.2 PyTorch: Tabular MLP (二分类)
# =========================
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader

device =torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("\n[PyTorch] device:", device)

# 准备 tensor (用 dense)
Xtr =torch.tensor(X_train_dense, dtype=torch.float32)
Xte =torch.tensor(X_test_dense, dtype=torch.float32)
ytr =torch.tensor(y_train.reshape(-1, 1), dtype=torch.float32)
yte =torch.tensor(y_test.reshape(-1, 1), dtype=torch.float32)

train_loader =DataLoader(TensorDataset(Xtr, ytr), batch_size=256, shuffle=True)
test_loader =DataLoader(TensorDataset(Xte, yte), batch_size=512, shuffle=False)

class TabularMLP(nn.Module):
    def __init__(self, in_dim):
        super().__init__()
        self.net =nn.Sequential(
            nn.Linear(in_dim, 512),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(256, 1) # logits
        )
    def forward(self, x):
        return self.net(x)

torch_model =TabularMLP(in_dim=X_train_dense.shape[1]).to(device)

# pos_weight 用于 BCEWithLogitsLoss: 正类权重 (越大越强调少数类)
pos_weight =torch.tensor([class_weight[1] /class_weight[0]], dtype=torch.float32).to(device)
criterion =nn.BCEWithLogitsLoss(pos_weight=pos_weight)
optimizer =torch.optim.AdamW(torch_model.parameters(), lr=1e-3, weight_decay=1e-4)

def torch_predict_proba(model, X_tensor, batch_size=1024):
    model.eval()
```

```
    probs =[]
    with torch.no_grad():
        for i in range(0, X_tensor.shape[0], batch_size):
            xb =X_tensor[i:i+batch_size].to(device)
            logits =model(xb)
            pb =torch.sigmoid(logits).cpu().numpy().reshape(-1)
            probs.append(pb)
    return np.concatenate(probs, axis=0)

# 训练若干 epoch
EPOCHS =20
best_f1 =-1
best_state =None

for epoch in range(1, EPOCHS +1):
    torch_model.train()
    total_loss =0.0
    for xb, yb in train_loader:
        xb, yb =xb.to(device), yb.to(device)
        optimizer.zero_grad()
        logits =torch_model(xb)
        loss =criterion(logits, yb)
        loss.backward()
        optimizer.step()
        total_loss +=loss.item() *xb.size(0)

    avg_loss =total_loss /len(train_loader.dataset)

    # 简单验证（在 test 上看 F1，做个 best checkpoint）
    y_prob =torch_predict_proba(torch_model, Xte)
    y_pred =(y_prob >=0.5).astype(int)
    f1 =f1_score(y_test, y_pred, zero_division=0)

    print(f"[Torch] Epoch {epoch:02d}/{EPOCHS} | loss={avg_loss:.4f} | test_f1={f1:.4f}")
    if f1 >best_f1:
        best_f1 =f1
        best_state ={k: v.detach().cpu().clone() for k, v in torch_model.state_dict().items()}

# 恢复最佳权重
if best_state is not None:
    torch_model.load_state_dict(best_state)

# 用 sklearn 风格评估 + 作图（复用上面的可视化逻辑）
class TorchWrapper:
    """把 PyTorch 模型包装成拥有 predict/predict_proba 接口的对象，复用评估函数。"""
```

```
    def __init__(self, model):
        self.model =model
    def predict_proba(self, X):
        Xt =torch.tensor(X, dtype=torch.float32)
        p =torch_predict_proba(self.model, Xt)
        # 返回 shape=(n,2)
        return np.stack([1 -p, p], axis=1)
    def predict(self, X):
        p =self.predict_proba(X)[:, 1]
        return (p >=0.5).astype(int)

tw =TorchWrapper(torch_model)
results.append(evaluate_binary_classifier(tw, X_train_dense, y_train, X_test_dense, y_test, "TorchMLP"))

# ==========================
# 11.3 汇总结果表（打印 + 可选保存 CSV）
# ==========================
res_df =pd.DataFrame(results).sort_values(by="f1", ascending=False)
print("\n===== Model Comparison (sorted by F1) =====")
print(res_df)

out_csv =os.path.join(fig_dir, "model_results_summary.csv")
res_df.to_csv(out_csv, index=False)
print("Saved:", out_csv)

# 可选：画一个横向条形图比较 F1 / AUC
plt.figure()
plt.barh(res_df["model"], res_df["f1"].values, edgecolor="black", linewidth=0.4)
plt.title("Model Comparison: F1 score")
plt.xlabel("F1")
plt.ylabel("Model")
save_show("compare_models_f1_barh.png")

plt.figure()
plt.barh(res_df["model"], res_df["roc_auc"].values, edgecolor="black", linewidth=0.4)
plt.title("Model Comparison: ROC-AUC")
plt.xlabel("ROC-AUC")
plt.ylabel("Model")
save_show("compare_models_rocauc_barh.png")
```