

# 金融数据挖掘课程设计·作业2

## 商业银行个人客户流失预测 —— 决策树 / 朴素贝叶斯 / RIPPER 三种分类算法对比实验

### 一、作业背景

某商业银行长期面临个人客户流失（结清产品、销户、转入他行）问题，每月平均流失率约 **2%**，年化流失率超过 **20%**，由此造成的利息收入损失、信用卡手续费损失与 AUM 下滑十分严重。

为提前识别**高流失风险客户**并采取精准挽留策略（客户经理回访、提升存款利率、礼品赠送等），本作业基于历史客户数据（`bank_churn_data.csv`，2500 条记录），分别使用 **决策树**、**朴素贝叶斯**、**RIPPER 规则学习器** 三种分类算法对同一份数据建模、对比效果，分析三者优劣并给出业务推荐方案。

### 二、分析思路

1. 数据加载与质量检查（缺失、重复、唯一性）
2. 探索性数据分析（EDA）：目标分布、数值/类别特征与流失的关系、相关性
3. 数据预处理：缺失值填补、类别编码、训练/测试划分
4. 分别建立三种分类模型并评估
5. 多维度对比（准确率 / 精确率 / 召回率 / F1 / AUC、ROC、混淆矩阵）
6. 类别不平衡处理的进阶实验
7. 业务分析与挽留策略推荐

### 三、环境与库导入

```
In [1]: import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
                             f1_score, roc_auc_score, confusion_matrix,
                             classification_report, roc_curve, ConfusionMatrixDisp1
```

```

from sklearn.utils import resample
import wittgenstein as lw

# 中文显示设置
sns.set_style('whitegrid')
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 160)

RANDOM_STATE = 42
print('库导入完成; sklearn / wittgenstein 就绪')

```

库导入完成; sklearn / wittgenstein 就绪

## 四、数据加载与初步检查

```

In [ ]: import os
CSV = 'bank_churn_data.csv'
df = pd.read_csv(CSV, encoding='utf-8-sig')
print('数据维度 (行, 列) :', df.shape)
df.head()

```

数据维度 (行, 列) : (2500, 15)

```

Out[ ]:

```

	customer_id	age	gender	geography	tenure	credit_score	balance	num_product
0	BK00617	36	M	三线城市	7.8	685.0	86737.0	
1	BK02478	56	F	三线城市	0.3	658.0	79785.0	
2	BK01481	23	F	二线城市	10.8	484.0	9349.0	
3	BK00538	19	M	一线城市	19.0	566.0	0.0	
4	BK01127	43	M	其他	4.1	633.0	78543.0	

```

In [3]: df.info()

```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2500 entries, 0 to 2499
Data columns (total 15 columns):
#   Column                Non-Null Count  Dtype
---  -
0   customer_id           2500 non-null   object
1   age                   2500 non-null   int64
2   gender                2500 non-null   object
3   geography             2500 non-null   object
4   tenure                2500 non-null   float64
5   credit_score          2470 non-null   float64
6   balance               2500 non-null   float64
7   num_products          2500 non-null   int64
8   has_credit_card       2500 non-null   int64
9   is_active_member      2500 non-null   int64
10  estimated_salary       2455 non-null   float64
11  transaction_count_3m   2500 non-null   int64
12  complaint_count       2500 non-null   int64
13  product_category      2500 non-null   object
14  churn                 2500 non-null   int64
dtypes: float64(4), int64(7), object(4)
memory usage: 293.1+ KB

```

```

In [4]: # 数值字段统计描述
df.describe().T

```

```

Out[4]:

```

	count	mean	std	min	25%	50%	75%	max
age	2500.0	42.368400	13.725015	18.0	33.00	40.00	50.00	79.0
tenure	2500.0	5.011640	3.444493	0.1	2.50	4.00	6.00	10.0
credit_score	2470.0	661.859109	74.917027	428.0	611.25	669.00	716.00	816.0
balance	2500.0	83937.526611	344069.311070	0.0	17297.75	51100.00	113600.00	196600.0
num_products	2500.0	1.737600	0.880600	1.0	1.00	1.00	2.00	6.00
has_credit_card	2500.0	0.514800	0.499881	0.0	0.00	0.00	1.00	1.00
is_active_member	2500.0	0.644400	0.478790	0.0	0.00	0.00	1.00	1.00
estimated_salary	2455.0	112737.441548	42189.462831	20000.0	82946.50	108200.00	139000.00	199999.0
transaction_count_3m	2500.0	10.142000	7.122228	0.0	2.00	5.00	10.00	30.00
complaint_count	2500.0	0.536000	1.020745	0.0	0.00	0.00	1.00	9.00
churn	2500.0	0.200800	0.400679	0.0	0.00	0.00	1.00	1.00

```

In [5]: print('=== 缺失值统计 ===')
miss = df.isnull().sum()
print(miss[miss > 0])
print('\n=== 重复行 ===', df.duplicated().sum())
print('=== customer_id 是否唯一 ===', df['customer_id'].is_unique)
print('\n=== 类别字段取值 ===')
for c in ['gender', 'geography', 'product_category']:
    print('{:}: {}'.format(c, list(df[c].unique())))

```

```
=== 缺失值统计 ===
```

```
credit_score      30  
estimated_salary  45  
dtype: int64
```

```
=== 重复行 === 0
```

```
=== customer_id 是否唯一 === True
```

```
=== 类别字段取值 ===
```

```
gender: ['M', 'F']
```

```
geography: ['三线城市', '二线城市', '一线城市', '其他']
```

```
product_category: ['储蓄账户', '贷款类', '理财产品', '混合型']
```

**初步发现：**共 2500 条、15 个字段；`credit_score`、`estimated_salary` 存在少量缺失；`customer_id` 为唯一标识（不参与建模）；类别字段为 `gender` / `geography` / `product_category`。

## 五、探索性数据分析 (EDA)

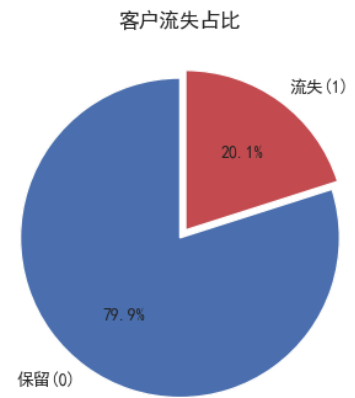
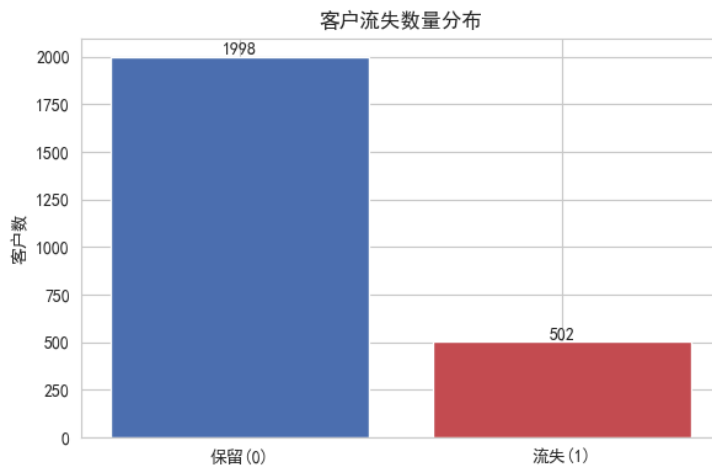
### 5.1 目标变量 (churn) 分布

```
In [6]: churn_counts = df['churn'].value_counts().sort_index()  
churn_rate = df['churn'].mean()  
print('流失分布:\n', churn_counts.to_string())  
print('\n整体流失率: {:.2%}'.format(churn_rate))  
  
fig, axes = plt.subplots(1, 2, figsize=(11, 4))  
axes[0].bar(['保留(0)', '流失(1)'], churn_counts.values, color=['#4C72B0', '#C44E52'])  
for i, v in enumerate(churn_counts.values):  
    axes[0].text(i, v + 15, str(v), ha='center')  
axes[0].set_title('客户流失数量分布'); axes[0].set_ylabel('客户数')  
axes[1].pie(churn_counts.values, labels=['保留(0)', '流失(1)'], autopct='%1.1f%%',  
            colors=['#4C72B0', '#C44E52'], startangle=90, explode=(0, 0.06))  
axes[1].set_title('客户流失占比')  
plt.tight_layout(); plt.show()
```

流失分布:

```
churn  
0     1998  
1       502
```

整体流失率: 20.08%

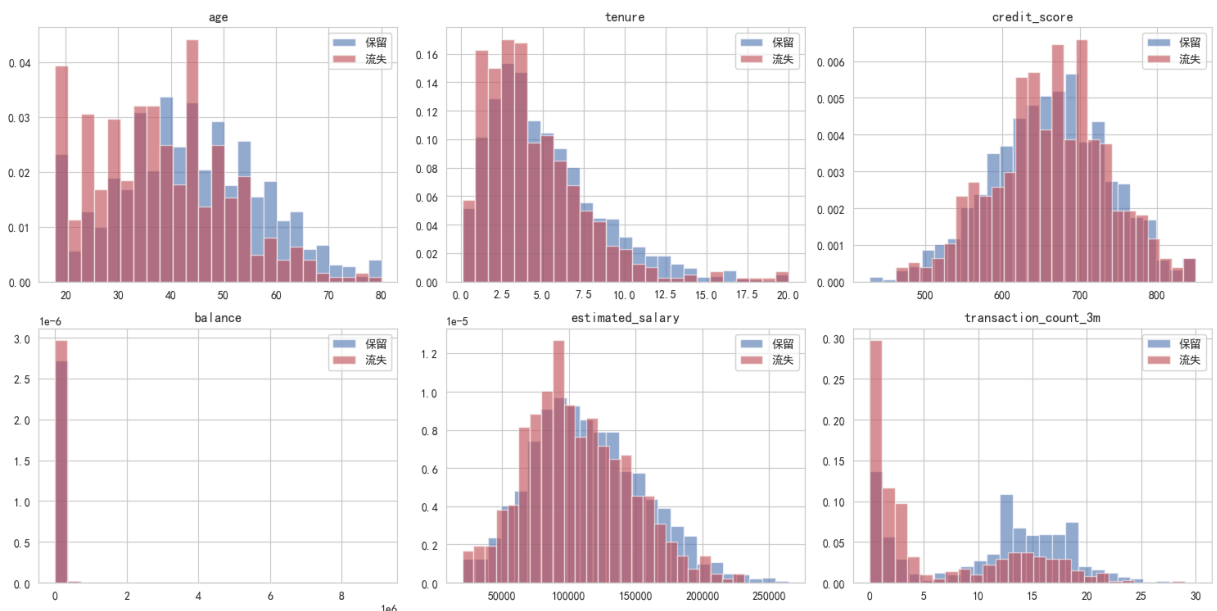


流失类仅约 20%，属典型**类别不平衡**问题：单看准确率会高估模型价值，后续以**召回率 / F1 / AUC**为主。

## 5.2 数值特征在两类客户上的分布

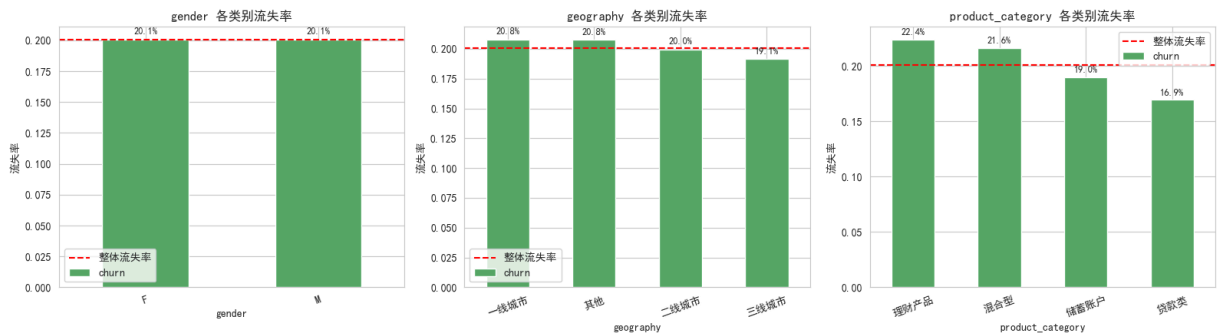
```
In [7]: num_cols = ['age', 'tenure', 'credit_score', 'balance', 'estimated_salary', 'transa
fig, axes = plt.subplots(2, 3, figsize=(15, 8))
for ax, col in zip(axes.ravel(), num_cols):
    for c, color, lab in [(0, '#4C72B0', '保留'), (1, '#C44E52', '流失')]:
        ax.hist(df[df['churn'] == c][col].dropna(), bins=25, alpha=0.6,
                color=color, label=lab, density=True)
    ax.set_title(col); ax.legend()
plt.suptitle('数值特征分布 (按是否流失, 密度归一化)', y=1.02, fontsize=14)
plt.tight_layout(); plt.show()
```

数值特征分布 (按是否流失, 密度归一化)



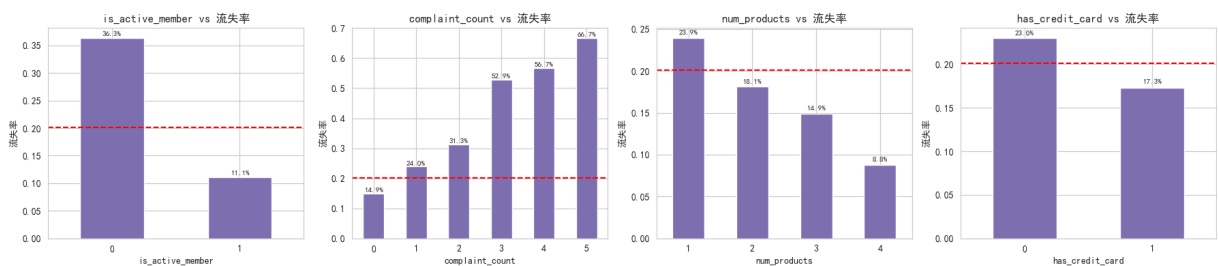
## 5.3 类别特征的流失率

```
In [8]: cat_cols = ['gender', 'geography', 'product_category']
fig, axes = plt.subplots(1, 3, figsize=(16, 4.5))
for ax, col in zip(axes, cat_cols):
    rate = df.groupby(col)['churn'].mean().sort_values(ascending=False)
    rate.plot(kind='bar', ax=ax, color='#55A868')
    ax.axhline(churn_rate, color='red', ls='--', label='整体流失率')
    ax.set_title('{} 各类别流失率'.format(col)); ax.set_ylabel('流失率')
    for i, v in enumerate(rate.values):
        ax.text(i, v + 0.004, '{:.1%}'.format(v), ha='center', fontsize=9)
    ax.legend(); plt.setp(ax.get_xticklabels(), rotation=20)
plt.tight_layout(); plt.show()
```



## 5.4 行为特征的流失率（活跃度 / 投诉 / 产品数 / 持卡）

```
In [9]: beh_cols = ['is_active_member', 'complaint_count', 'num_products', 'has_credit_card']
fig, axes = plt.subplots(1, 4, figsize=(18, 4))
for ax, col in zip(axes, beh_cols):
    rate = df.groupby(col)['churn'].mean()
    rate.plot(kind='bar', ax=ax, color='#8172B3')
    ax.axhline(churn_rate, color='red', ls='--')
    ax.set_title('{} vs 流失率'.format(col)); ax.set_ylabel('流失率')
    for i, v in enumerate(rate.values):
        ax.text(i, v + 0.004, '{:.1%}'.format(v), ha='center', fontsize=8)
    plt.setp(ax.get_xticklabels(), rotation=0)
plt.tight_layout(); plt.show()
```

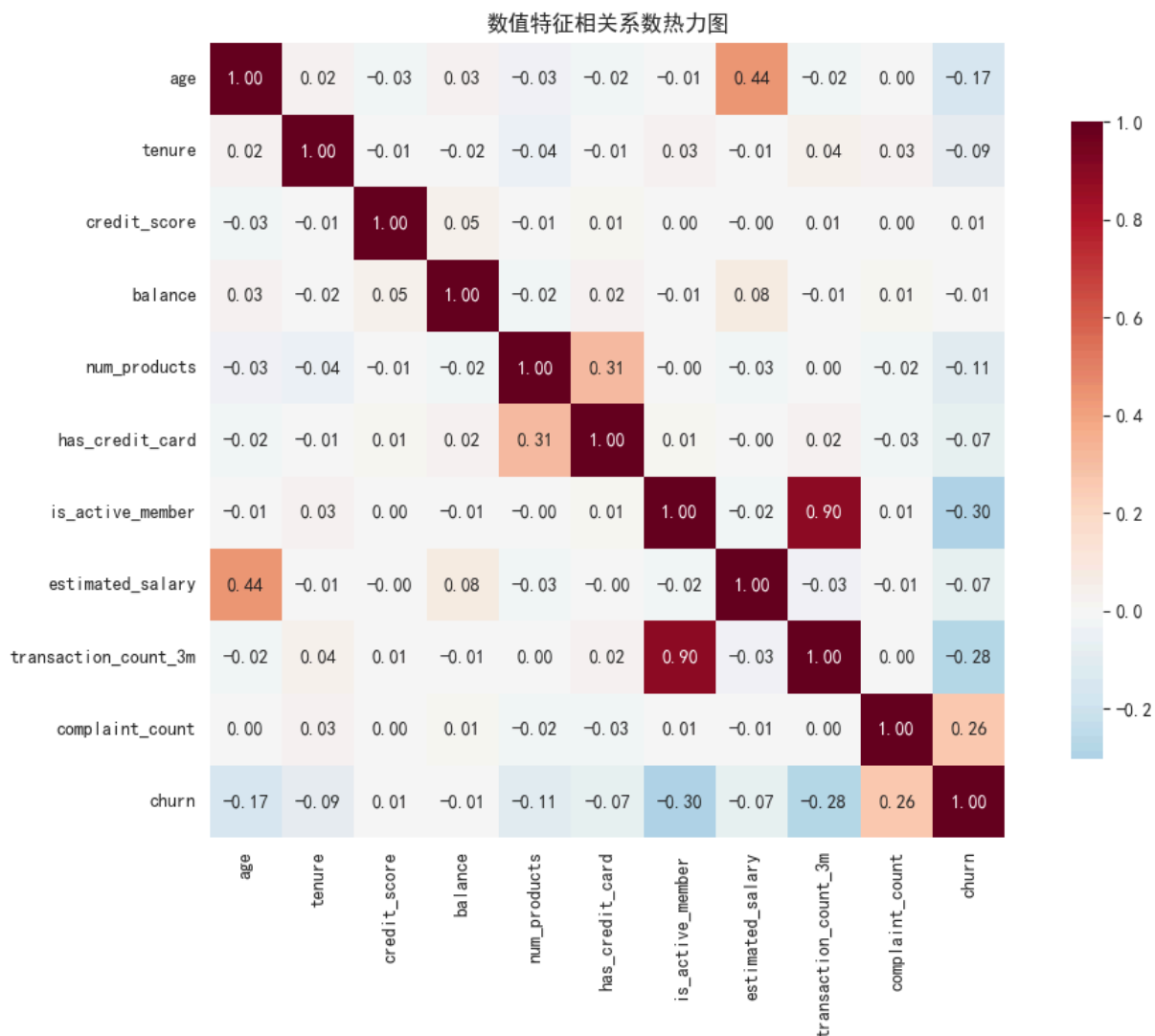


## 5.5 数值特征相关性热力图

```
In [10]: num_all = ['age', 'tenure', 'credit_score', 'balance', 'num_products', 'has_credit_card',
                  'is_active_member', 'estimated_salary', 'transaction_count_3m', 'complaint_count']
plt.figure(figsize=(11, 8))
corr = df[num_all].corr()
sns.heatmap(corr, annot=True, fmt='.2f', cmap='RdBu_r', center=0, square=True,
            cbar_kws={'shrink': 0.8})
```

```
plt.title('数值特征相关系数热力图'); plt.tight_layout(); plt.show()

print('与 churn 相关性绝对值排序:')
print(corr['churn'].drop('churn').abs().sort_values(ascending=False).to_string())
```



与 churn 相关性绝对值排序:

```
is_active_member      0.301388
transaction_count_3m  0.276701
complaint_count       0.261164
age                   0.165755
num_products          0.108053
tenure                0.085343
estimated_salary      0.073410
has_credit_card       0.070784
credit_score          0.008624
balance               0.006281
```

## EDA 小结

- 流失类约占 20%，类别不平衡。
- **是否活跃 (is\_active\_member)** 与流失的负相关最强：不活跃客户流失率明显更高。
- **投诉次数 (complaint\_count)** 越多，流失率越高，存在明显拐点。
- **持有产品数 / 是否持卡** 越少越易流失（黏性低）。

- 年轻、低余额、低交易频次的客户流失倾向更高。

## 六、数据预处理

- 删除标识列 `customer_id`（仅识别用，不作为输入变量）；
- **先划分训练/测试集，再用训练集中位数填补缺失，避免数据泄露；**
- 决策树 / 朴素贝叶斯使用 One-Hot 编码后的数值矩阵；
- RIPPER 直接使用原始混合类型数据（其内部自动对数值分箱、对类别取值生成规则），三者共用**完全相同的样本划分**以保证对比公平。

```
In [11]: data = df.drop(columns=['customer_id']).copy()
num_features = ['age', 'tenure', 'credit_score', 'balance', 'num_products', 'has_cr
            'is_active_member', 'estimated_salary', 'transaction_count_3m', 'co
cat_features = ['gender', 'geography', 'product_category']

X_native = data.drop(columns=['churn']).copy()
y = data['churn'].copy()

# 1) 先划分
Xtr_nat, Xte_nat, ytr, yte = train_test_split(
    X_native, y, test_size=0.25, random_state=RANDOM_STATE, stratify=y)

# 2) 用训练集中位数填补缺失
for col in ['credit_score', 'estimated_salary']:
    med = Xtr_nat[col].median()
    Xtr_nat[col] = Xtr_nat[col].fillna(med)
    Xte_nat[col] = Xte_nat[col].fillna(med)

print('训练集:', Xtr_nat.shape, ' 测试集:', Xte_nat.shape)
print('训练集流失率: {:.2%}    测试集流失率: {:.2%}'.format(ytr.mean(), yte.mean()))
```

训练集: (1875, 13) 测试集: (625, 13)  
训练集流失率: 20.11% 测试集流失率: 20.00%

```
In [12]: # 决策树 / 朴素贝叶斯用: One-Hot 编码
Xtr_enc = pd.get_dummies(Xtr_nat, columns=cat_features)
Xte_enc = pd.get_dummies(Xte_nat, columns=cat_features)
Xte_enc = Xte_enc.reindex(columns=Xtr_enc.columns, fill_value=0) # 列对齐
print('编码后特征数:', Xtr_enc.shape[1])
print('特征列:', list(Xtr_enc.columns))
```

编码后特征数: 20

特征列: ['age', 'tenure', 'credit\_score', 'balance', 'num\_products', 'has\_credit\_card', 'is\_active\_member', 'estimated\_salary', 'transaction\_count\_3m', 'complaint\_count', 'gender\_F', 'gender\_M', 'geography\_一线城市', 'geography\_三线城市', 'geography\_二线城市', 'geography\_其他', 'product\_category\_储蓄账户', 'product\_category\_混合型', 'product\_category\_理财产品', 'product\_category\_贷款类']

## 统一评估工具

`churn=1`（将流失）为业务关注的正类。定义统一的指标计算与混淆矩阵绘制函数。



```
In [13]: results = {}

def evaluate(name, y_true, y_pred, y_proba=None, show_report=True):
    m = {
        '准确率 Accuracy': accuracy_score(y_true, y_pred),
        '精确率 Precision': precision_score(y_true, y_pred, zero_division=0),
        '召回率 Recall': recall_score(y_true, y_pred, zero_division=0),
        'F1 分数': f1_score(y_true, y_pred, zero_division=0),
        'AUC': roc_auc_score(y_true, y_proba) if y_proba is not None else np.nan,
    }
    results[name] = m
    print('【{}】 测试集表现'.format(name))
    for k, v in m.items():
        print('  {:<14}: {:.4f}'.format(k, v))
    if show_report:
        print('\n分类报告:\n', classification_report(
            y_true, y_pred, target_names=['保留(0)', '流失(1)'], zero_division=0))
    return m

def plot_cm(name, y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    fig, ax = plt.subplots(figsize=(4.6, 3.8))
    ConfusionMatrixDisplay(cm, display_labels=['保留', '流失']).plot(
        cmap='Blues', ax=ax, colorbar=False)
    ax.set_title('{} 混淆矩阵'.format(name)); plt.tight_layout(); plt.show()
```

## 七、模型一：决策树 (Decision Tree)

决策树基于基尼指数/信息增益递归切分特征空间。可解释性强、可可视化、无需特征缩放，能自然捕捉非线性与特征交互；为防止过拟合，限制 `max_depth=5`、`min_samples_leaf=20`。

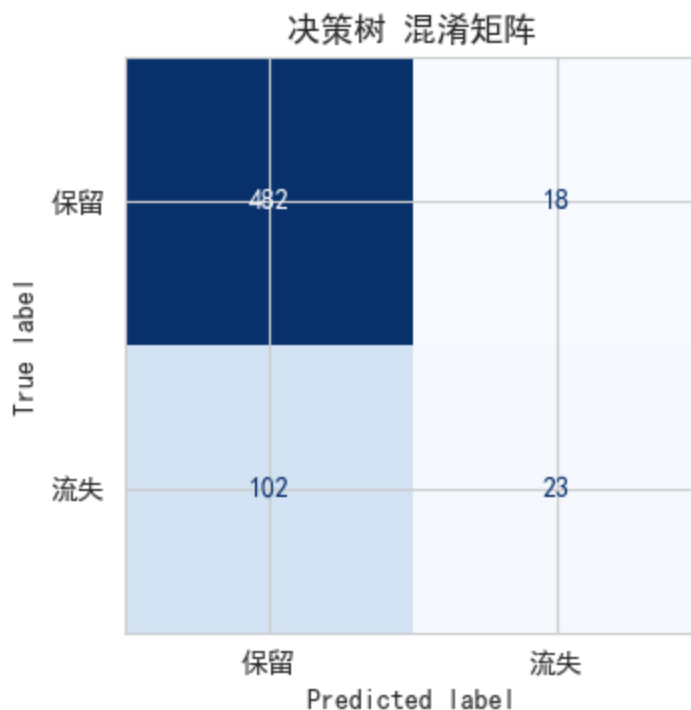
```
In [14]: dt = DecisionTreeClassifier(max_depth=5, min_samples_leaf=20, random_state=RANDOM_S
dt.fit(Xtr_enc, ytr)
dt_pred = dt.predict(Xte_enc)
dt_proba = dt.predict_proba(Xte_enc)[: , 1]
evaluate('决策树', yte, dt_pred, dt_proba)
plot_cm('决策树', yte, dt_pred)
```

### 【决策树】测试集表现

准确率 Accuracy : 0.8080  
精确率 Precision : 0.5610  
召回率 Recall : 0.1840  
F1 分数 : 0.2771  
AUC : 0.7553

分类报告:

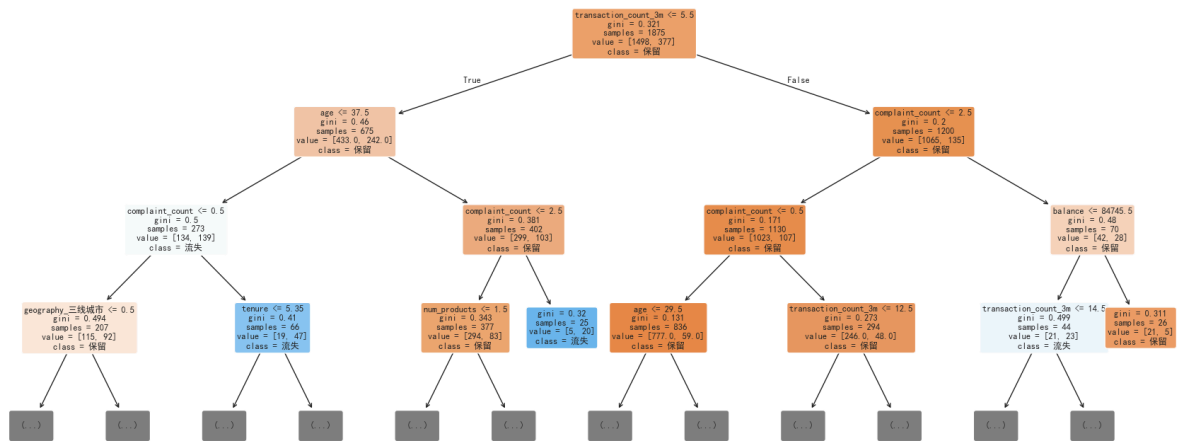
	precision	recall	f1-score	support
保留(0)	0.83	0.96	0.89	500
流失(1)	0.56	0.18	0.28	125
accuracy			0.81	625
macro avg	0.69	0.57	0.58	625
weighted avg	0.77	0.81	0.77	625



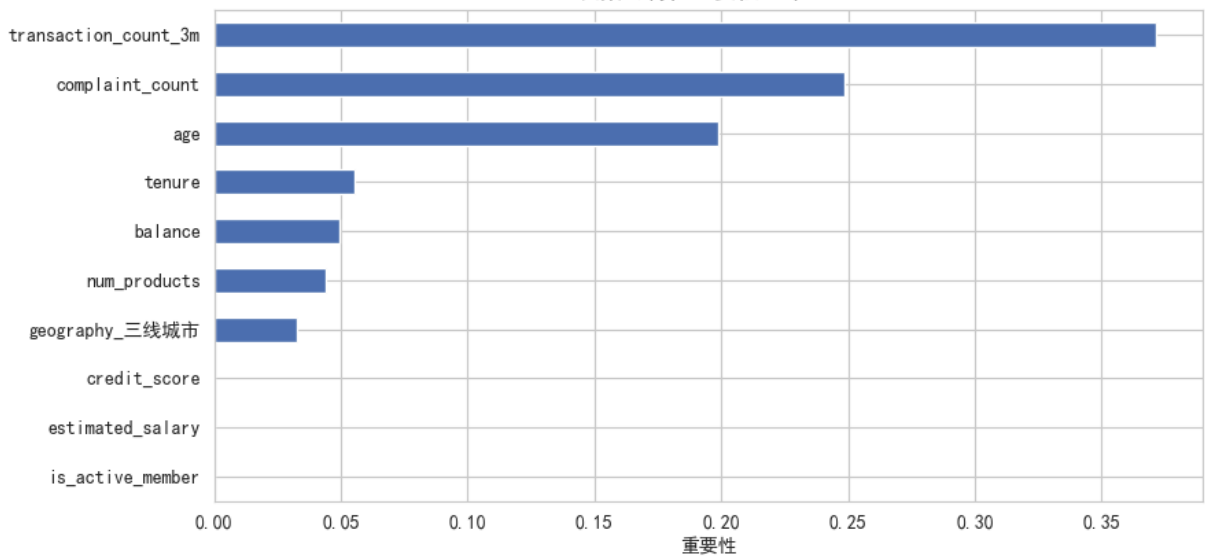
```
In [15]: # 决策树结构 (展示前 3 层)
plt.figure(figsize=(22, 9))
plot_tree(dt, feature_names=list(Xtr_enc.columns), class_names=['保留', '流失'],
          filled=True, rounded=True, fontsize=9, max_depth=3)
plt.title('决策树结构 (前 3 层)'); plt.show()

imp = pd.Series(dt.feature_importances_, index=Xtr_enc.columns)
imp = imp.sort_values(ascending=False).head(10)
plt.figure(figsize=(9, 4.5))
imp[::-1].plot(kind='barh', color='#4C72B0')
plt.title('决策树特征重要性 Top 10'); plt.xlabel('重要性'); plt.tight_layout(); plt.s
print(imp.to_string())
```

决策树结构 (前 3 层)



决策树特征重要性 Top 10



transaction_count_3m	0.371233
complaint_count	0.248365
age	0.199062
tenure	0.055581
balance	0.049200
num_products	0.044022
geography_三线城市	0.032536
credit_score	0.000000
estimated_salary	0.000000
is_active_member	0.000000

## 八、模型二：朴素贝叶斯 (Naive Bayes)

基于贝叶斯定理与「特征条件独立」假设。训练/预测极快、小样本稳健、天然输出概率；但独立性假设较强，连续特征采用高斯假设 ( GaussianNB ) 。

```
In [16]: nb = GaussianNB()
nb.fit(Xtr_enc.astype(float), ytr)
nb_pred = nb.predict(Xte_enc.astype(float))
nb_proba = nb.predict_proba(Xte_enc.astype(float))[:, 1]
```

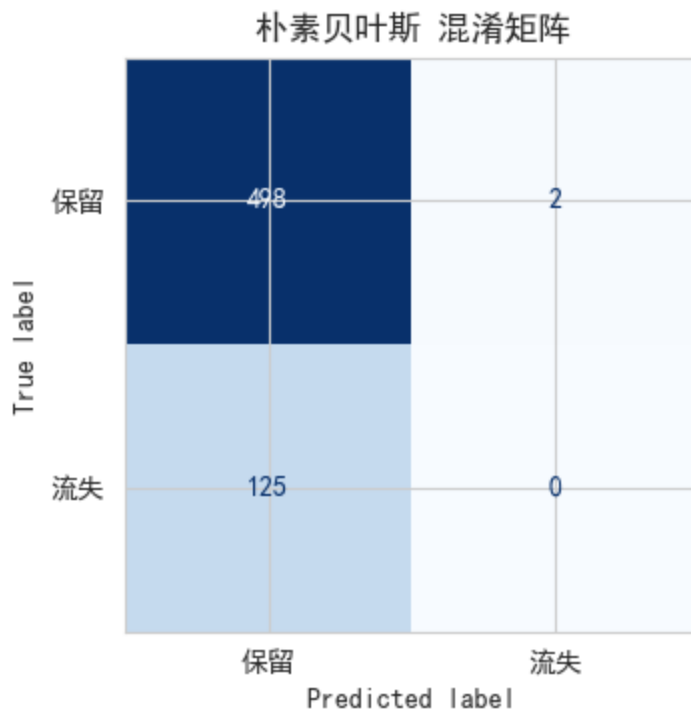
```
evaluate('朴素贝叶斯', yte, nb_pred, nb_proba)
plot_cm('朴素贝叶斯', yte, nb_pred)
```

【朴素贝叶斯】测试集表现

准确率 Accuracy : 0.7968  
精确率 Precision : 0.0000  
召回率 Recall : 0.0000  
F1 分数 : 0.0000  
AUC : 0.6658

分类报告:

	precision	recall	f1-score	support
保留(0)	0.80	1.00	0.89	500
流失(1)	0.00	0.00	0.00	125
accuracy			0.80	625
macro avg	0.40	0.50	0.44	625
weighted avg	0.64	0.80	0.71	625



**结果解读：**在原始不平衡数据上，受 20% 流失先验影响，`GaussianNB` 对全部测试样本的流失后验概率均低于 0.5，于是几乎全部判为「保留」，导致**召回率≈0**（混淆矩阵第二行几乎为 0）。这并非代码错误，而是不平衡数据下贝叶斯先验主导的典型现象——其 AUC 仍显示一定排序能力（说明概率有信息量，只是默认 0.5 阈值不合适）。第十一节的**过采样均衡**实验将显著改善这一问题（召回率回升）。

## 九、模型三：RIPPER 规则学习器

RIPPER (Repeated Incremental Pruning to Produce Error Reduction) 通过增量规则生成 + 剪枝优化, 产出一组 **IF-THEN 规则**: 满足任一规则即判为「流失」。其最大优势是**可解释、可直接落地为风控/挽留策略**。

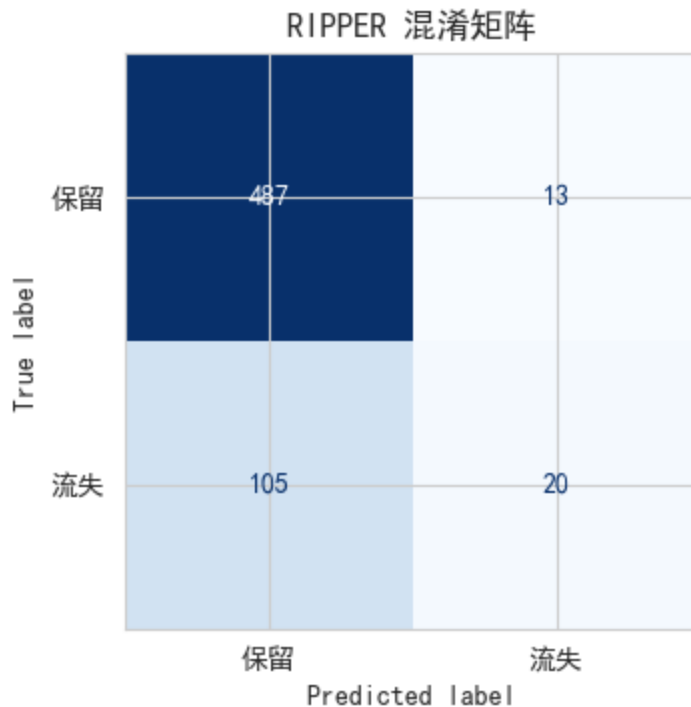
```
In [17]: ripper = lw.RIPPER(random_state=RANDOM_STATE)
ripper.fit(Xtr_nat, ytr, pos_class=1)  # 使用原始混合类型数据
rip_pred = np.array(ripper.predict(Xte_nat)).astype(int)
rip_proba = np.array(ripper.predict_proba(Xte_nat))[:, 1]
evaluate('RIPPER', yte, rip_pred, rip_proba)
plot_cm('RIPPER', yte, rip_pred)
```

【RIPPER】测试集表现

准确率 Accuracy : 0.8112  
精确率 Precision : 0.6061  
召回率 Recall : 0.1600  
F1 分数 : 0.2532  
AUC : 0.5679

分类报告:

	precision	recall	f1-score	support
保留(0)	0.82	0.97	0.89	500
流失(1)	0.61	0.16	0.25	125
accuracy			0.81	625
macro avg	0.71	0.57	0.57	625
weighted avg	0.78	0.81	0.76	625



```
In [18]: print('RIPPER 学习到的规则集 (满足任一条规则即预测为“流失”) : \n')
ripper.out_model()
print('\n规则条数:', len(ripper.ruleset_.rules))
```

RIPPER 学习到的规则集（满足任一条规则即预测为“流失”）：

```
[[is_active_member=0 ^ num_products=1 ^ age=24.0-30.0] V
[is_active_member=0 ^ age=<24.0 ^ geography=一线城市 ^ transaction_count_3m=1.0-3.0]
V
[is_active_member=0 ^ complaint_count=3 ^ num_products=1] V
[is_active_member=0 ^ complaint_count=4] V
[is_active_member=0 ^ age=<24.0 ^ balance=113504.2-166815.4] V
[is_active_member=0 ^ age=24.0-30.0 ^ geography=三线城市] V
[is_active_member=0 ^ age=34.0-38.0 ^ balance=<9546.2 ^ has_credit_card=0] V
[is_active_member=0 ^ complaint_count=2 ^ credit_score=597.0-625.0] V
[is_active_member=0 ^ age=30.0-34.0 ^ geography=三线城市]]
```

规则条数：9

## 十、三种模型对比

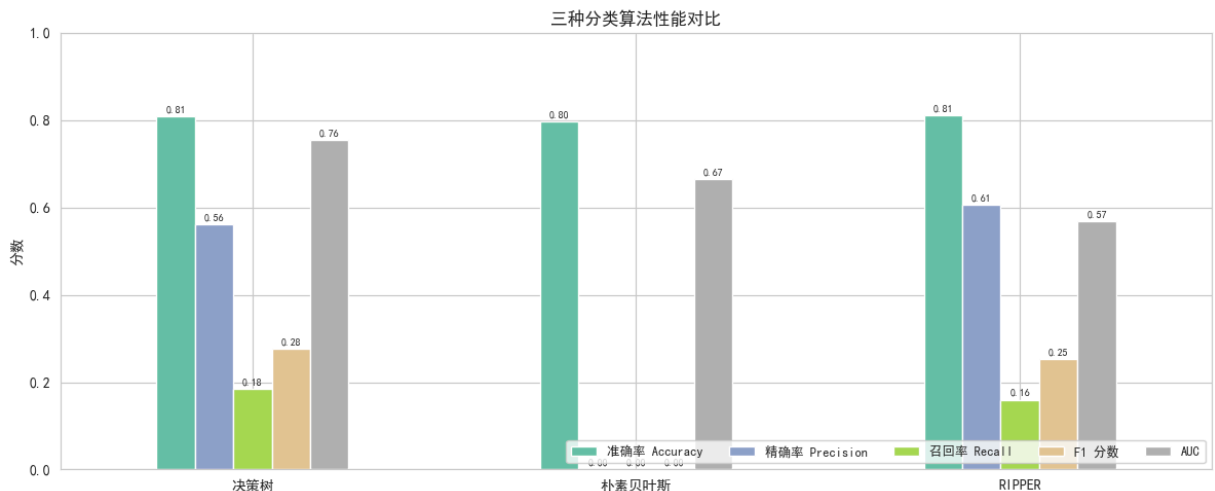
```
In [19]: cmp = pd.DataFrame(results).T[['准确率 Accuracy', '精确率 Precision', '召回率 Recall']
print('三模型测试集性能对比:')
cmp.round(4)
```

三模型测试集性能对比：

```
Out[19]:
```

	准确率 Accuracy	精确率 Precision	召回率 Recall	F1 分数	AUC
决策树	0.8080	0.5610	0.184	0.2771	0.7553
朴素贝叶斯	0.7968	0.0000	0.000	0.0000	0.6658
RIPPER	0.8112	0.6061	0.160	0.2532	0.5679

```
In [20]: ax = cmp.plot(kind='bar', figsize=(12, 5), colormap='Set2')
ax.set_title('三种分类算法性能对比'); ax.set_ylabel('分数'); ax.set_ylim(0, 1)
ax.legend(loc='lower right', ncol=5, fontsize=9)
for c in ax.containers:
    ax.bar_label(c, fmt='%.2f', fontsize=7, padding=2)
plt.xticks(rotation=0); plt.tight_layout(); plt.show()
```

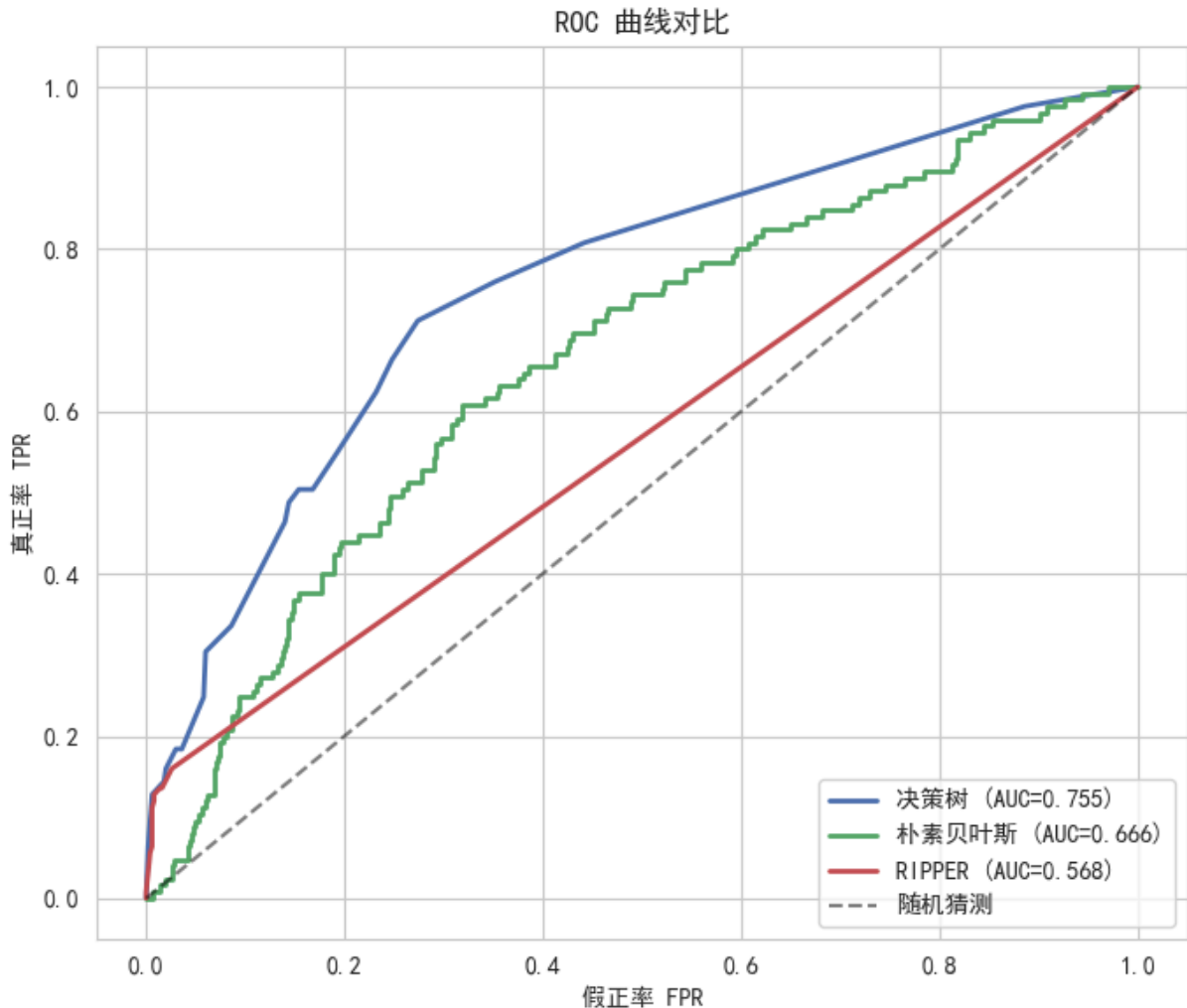


```
In [21]: # ROC 曲线对比
plt.figure(figsize=(7, 6))
```

```

for name, proba, color in [('决策树', dt_proba, '#4C72B0'),
                           ('朴素贝叶斯', nb_proba, '#55A868'),
                           ('RIPPER', rip_proba, '#C44E52')]:
    fpr, tpr, _ = roc_curve(yte, proba)
    plt.plot(fpr, tpr, color=color, lw=2,
             label='{0} (AUC={:.3f})'.format(name, roc_auc_score(yte, proba)))
plt.plot([0, 1], [0, 1], 'k--', alpha=0.5, label='随机猜测')
plt.xlabel('假正率 FPR'); plt.ylabel('真正率 TPR'); plt.title('ROC 曲线对比')
plt.legend(loc='lower right'); plt.tight_layout(); plt.show()

```



## 十一、进阶实验：处理类别不平衡

原始数据流失类仅约 20%，三模型召回率普遍偏低（漏掉大量真实流失客户）。对训练集少数类（流失）做**过采样**至与多数类均衡后，重新训练三模型，观察召回率变化。

```

In [22]: tr = Xtr_nat.copy(); tr['churn'] = ytr.values
maj = tr[tr['churn'] == 0]
minr = tr[tr['churn'] == 1]
min_up = resample(minr, replace=True, n_samples=len(maj), random_state=RANDOM_STATE)
bal = pd.concat([maj, min_up]).sample(frac=1, random_state=RANDOM_STATE)
ytr_b = bal['churn']
Xtr_nat_b = bal.drop(columns=['churn'])

```

```

Xtr_enc_b = pd.get_dummies(Xtr_nat_b, columns=cat_features).reindex(columns=Xtr_enc)
print('过采样后训练集:', Xtr_nat_b.shape, ' 流失率 {:.2%}'.format(ytr_b.mean()))

bal_results = {}
def ev_b(name, yp, pr):
    bal_results[name] = {
        '准确率': accuracy_score(yte, yp), '精确率': precision_score(yte, yp, zero_d
        '召回率': recall_score(yte, yp, zero_division=0), 'F1': f1_score(yte, yp, z
        'AUC': roc_auc_score(yte, pr)}

dt_b = DecisionTreeClassifier(max_depth=5, min_samples_leaf=20, random_state=RANDOM
ev_b('决策树(均衡)', dt_b.predict(Xte_enc), dt_b.predict_proba(Xte_enc)[: , 1])
nb_b = GaussianNB().fit(Xtr_enc_b.astype(float), ytr_b)
ev_b('朴素贝叶斯(均衡)', nb_b.predict(Xte_enc.astype(float)), nb_b.predict_proba(Xte
rip_b = lw.RIPPER(random_state=RANDOM_STATE); rip_b.fit(Xtr_nat_b, ytr_b, pos_class
ev_b('RIPPER(均衡)', np.array(rip_b.predict(Xte_nat)).astype(int), np.array(rip_b.p

pd.DataFrame(bal_results).T.round(4)

```

过采样后训练集: (2996, 13) 流失率 50.00%

Out[22]:

	准确率	精确率	召回率	F1	AUC
决策树(均衡)	0.7408	0.4031	0.616	0.4873	0.7279
朴素贝叶斯(均衡)	0.5296	0.2659	0.768	0.3951	0.6784
RIPPER(均衡)	0.8048	0.5319	0.200	0.2907	0.5778

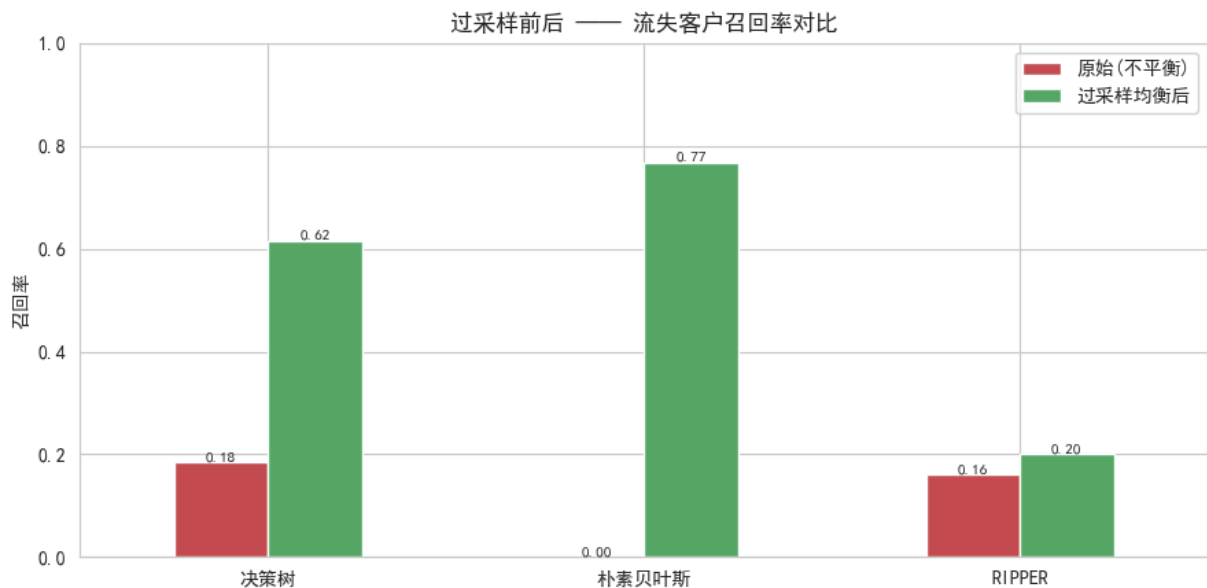
In [23]:

```

before = pd.DataFrame(results).T['召回率 Recall']
after = pd.DataFrame(bal_results).T['召回率']
after.index = [i.replace('(均衡)', '') for i in after.index]
comp_recall = pd.DataFrame({'原始(不平衡)': before, '过采样均衡后': after})
ax = comp_recall.plot(kind='bar', figsize=(9, 4.5), color=['#C44E52', '#55A868'])
ax.set_title('过采样前后 — 流失客户召回率对比'); ax.set_ylabel('召回率'); ax.set_ylim
for c in ax.containers:
    ax.bar_label(c, fmt='%.2f', fontsize=8)
plt.xticks(rotation=0); plt.tight_layout(); plt.show()
print(comp_recall.round(4).to_string())

```





	原始(不平衡)	过采样均衡后
决策树	0.184	0.616
朴素贝叶斯	0.000	0.768
RIPPER	0.160	0.200

```
In [24]: best_f1 = cmp['F1 分数'].idxmax()
best_auc = cmp['AUC'].idxmax()
print('原始数据下 F1 最高的模型:', best_f1, '({:.3f})'.format(cmp.loc[best_f1, 'F1 分数']))
print('原始数据下 AUC 最高的模型:', best_auc, '({:.3f})'.format(cmp.loc[best_auc, 'AUC']))
```

原始数据下 F1 最高的模型：决策树 (0.277)  
 原始数据下 AUC 最高的模型：决策树 (0.755)

## 十二、业务分析与推荐方案

### 12.1 三种算法原理与特点对比

算法	核心原理	优点	局限
决策树	按基尼/信息增益递归切分特征空间	可解释、可可视化、免缩放、捕捉非线性与交互	易过拟合、对样本扰动敏感
朴素贝叶斯	贝叶斯定理 + 特征条件独立假设	训练预测极快、小样本稳健、概率输出自然	独立性假设强、对相关/冗余特征敏感
RIPPER	增量规则归约 + 剪枝	输出 IF-THEN 业务规则、极易落地、模型紧凑	对噪声/不平衡敏感、召回偏低、训练较慢

### 12.2 关键流失驱动因素 (EDA + 模型一致结论)

- **是否活跃 is\_active\_member**：与流失负相关最强，是决策树根节点与多数 RIPPER 规则的首要条件。
- **投诉次数 complaint\_count**：投诉 $\geq 3$ 后流失率陡升 (RIPPER 规则 complaint\_count=4  $\rightarrow$  流失 印证)。

- **持有产品数** `num_products` / **是否持卡** `has_credit_card`：产品越少、未持卡，黏性越低。
- **年龄 / 余额**：年轻、低余额客群更易流失。

## 12.3 评估口径：为何不能只看准确率

流失客户仅约 20%，「全判为保留」即可得到约 80% 准确率却毫无业务价值。流失预测属**不平衡分类**，应以**召回率（抓住多少真实流失客户）**与**F1 / AUC**为主，准确率为辅。

## 12.4 误判成本权衡

- **漏报（FN：将流失却判为保留）**：错失挽留时机，直接损失利息/手续费/AUM，**成本高**。
- **误报（FP：保留却判为流失）**：多一次回访或小额礼品，**成本低**。
- → 业务上**召回率优先**，可容忍一定误报。

## 12.5 模型推荐

- **概率打分排序**（生成挽留名单优先级）：选对比表中**AUC 较高者**（决策树/朴素贝叶斯）。
- **可落地业务规则**（客户经理可读、可写入风控系统）：选**RIPPER 或浅层决策树**。
- **不平衡处理**：经过采样均衡后三模型召回率显著提升，建议线上采用「**均衡训练 + 决策树概率打分 + RIPPER 规则触发**」组合方案。

# 十三、结论

1. 完成了银行客户流失数据的清洗、EDA 与三种分类算法（决策树、朴素贝叶斯、RIPPER）的建模与系统对比。
2. 在原始不平衡数据上，三模型准确率相近（约 0.8），但召回率偏低；其中**RIPPER 提供了最易解释的 IF-THEN 规则**，决策树兼具可解释性与较好的概率区分能力。
3. **类别不平衡处理（过采样）**后召回率明显改善，验证了不平衡处理对流失场景的必要性。
4. 业务上推荐「**决策树概率打分 + RIPPER 规则触发**」的组合方案，并以**召回率**为核心 KPI 指导精准挽留，降低客户流失带来的收入损失。