

个贷违约预测

本实验利用个人贷款违约记录数据 (data.csv) , 进行个贷违约预测。

主要步骤: 数据预处理 → 数据集划分 → 决策树 / 逻辑回归 / 朴素贝叶斯三种模型训练与评估 → 参数选择与效果比较。

1. 导入库

```
In [27]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import warnings
warnings.filterwarnings('ignore')

# 中文显示
matplotlib.rcParams['font.sans-serif'] = ['SimHei']
matplotlib.rcParams['axes.unicode_minus'] = False

from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
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
)

print('库导入完成')
```

库导入完成

2. 数据加载与初步探索

```
In [28]: df = pd.read_csv('data.csv')
print('数据形状:', df.shape)
df.head()
```

数据形状: (10001, 22)

```
Out[28]:
```

	loan_id	user_id	total_loan	year_of_loan	interest	monthly_payment	class	ei
0	1040418	240418.0	31818.18182	3.0	11.466	1174.91	C	
1	1025197	225197.0	28000.00000	5.0	16.841	670.69	C	
2	1009360	209360.0	17272.72727	3.0	8.900	603.32	A	
3	1039708	239708.0	20000.00000	3.0	4.788	602.30	A	
4	1027483	227483.0	15272.72727	3.0	12.790	470.31	C	

5 rows × 22 columns

```
In [29]: print('各字段类型与非空数量: ')
df.info()
```

各字段类型与非空数量:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10001 entries, 0 to 10000
Data columns (total 22 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   loan_id                               10001 non-null  object
1   user_id                               10000 non-null  float64
2   total_loan                            10000 non-null  float64
3   year_of_loan                          10000 non-null  float64
4   interest                              10000 non-null  float64
5   monthly_payment                       10000 non-null  float64
6   class                                 9991 non-null   object
7   employer_type                         9991 non-null   object
8   industry                              9982 non-null   object
9   work_year                             9373 non-null   object
10  house_exist                           10000 non-null  float64
11  issue_date                            10000 non-null  object
12  use                                    10000 non-null  float64
13  debt_loan_ratio                       10000 non-null  float64
14  del_in_18month                        10000 non-null  float64
15  known_outstanding_loan                10000 non-null  float64
16  known_dero                            10000 non-null  float64
17  pub_dero_bankrup                      9993 non-null   float64
18  early_return                          10000 non-null  float64
19  early_return_amount                   10000 non-null  float64
20  early_return_amount_3mon              10000 non-null  float64
21  isDefault                             10000 non-null  float64
dtypes: float64(16), object(6)
memory usage: 1.7+ MB
```

```
In [30]: print('基本统计描述: ')
df.describe()
```

基本统计描述:

Out[30]:

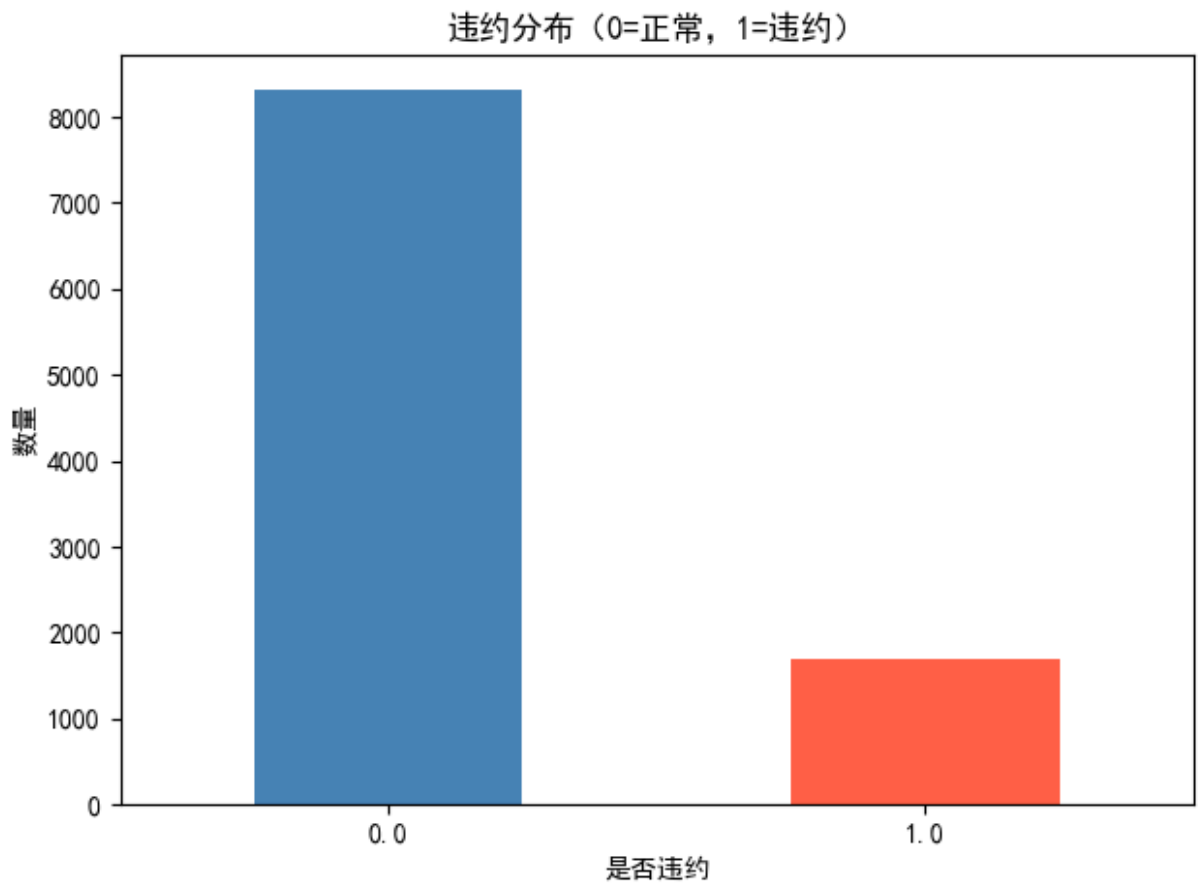
	user_id	total_loan	year_of_loan	interest	monthly_payment
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	225209.587700	14402.126591	3.479600	13.222782	436.960427
std	14386.820956	8953.946807	0.853965	4.875755	261.754396
min	200008.000000	818.181818	3.000000	4.779000	30.440000
25%	212973.250000	7500.000000	3.000000	9.702000	248.820000
50%	225276.500000	12272.727270	3.000000	12.639000	371.525000
75%	237694.500000	19636.363640	3.000000	15.985500	573.830000
max	249997.000000	47272.727270	5.000000	33.979000	1503.890000

In [31]:

```
print('目标变量 isDefault 分布: ')
print(df['isDefault'].value_counts())
print('违约率: {:.2%}'.format(df['isDefault'].mean()))

df['isDefault'].value_counts().plot(kind='bar', color=['steelblue', 'tomato'])
plt.title('违约分布 (0=正常, 1=违约) ')
plt.xlabel('是否违约')
plt.ylabel('数量')
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```

目标变量 isDefault 分布:
isDefault
0.0 8317
1.0 1683
Name: count, dtype: int64
违约率: 16.83%



3. 数据预处理

3.1 缺失值处理

```
In [32]: print('各列缺失值数量: ')  
missing = df.isnull().sum()  
print(missing[missing > 0])
```

各列缺失值数量：

user_id	1
total_loan	1
year_of_loan	1
interest	1
monthly_payment	1
class	10
employer_type	10
industry	19
work_year	628
house_exist	1
issue_date	1
use	1
debt_loan_ratio	1
del_in_18month	1
known_outstanding_loan	1
known_dero	1
pub_dero_bankrup	8
early_return	1
early_return_amount	1
early_return_amount_3mon	1
isDefault	1
dtype: int64	

```
In [33]: # 数值型列用中位数填充，类别型列用众数填充
for col in df.columns:
    if df[col].isnull().sum() > 0:
        if df[col].dtype == 'object':
            df[col].fillna(df[col].mode()[0], inplace=True)
        else:
            df[col].fillna(df[col].median(), inplace=True)

print('缺失值处理后，剩余缺失值总数：', df.isnull().sum().sum())
```

缺失值处理后，剩余缺失值总数： 0

3.2 特征工程

```
In [34]: # 删除无意义的 ID 列
df.drop(columns=['loan_id', 'user_id'], inplace=True)

# issue_date: 提取月份数字
df['issue_date'] = pd.to_datetime(df['issue_date'], errors='coerce')
df['issue_month'] = df['issue_date'].dt.month
df['issue_year'] = df['issue_date'].dt.year
df.drop(columns=['issue_date'], inplace=True)

# work_year: 将工作年限转换为数值
work_year_map = {
    '< 1 year': 0,
    '1 year': 1,
    '2 years': 2,
    '3 years': 3,
    '4 years': 4,
    '5 years': 5,
```

```

    '6 years': 6,
    '7 years': 7,
    '8 years': 8,
    '9 years': 9,
    '10+ years': 10
}
df['work_year'] = df['work_year'].map(work_year_map)
df['work_year'].fillna(df['work_year'].median(), inplace=True)

print('work_year 处理后样本: ')
print(df['work_year'].value_counts().sort_index())

```

work_year 处理后样本:

```

work_year
0      764
1      671
2      847
3      776
4      562
5      623
6      476
7      436
8      456
9      393
10     3997
Name: count, dtype: int64

```

```

In [35]: # 对剩余的类别型变量进行 LabelEncoder 编码
cat_cols = df.select_dtypes(include='object').columns.tolist()
print('需要编码的类别列: ', cat_cols)

le = LabelEncoder()
for col in cat_cols:
    df[col] = le.fit_transform(df[col].astype(str))

print('编码完成, 数据形状: ', df.shape)
df.head(3)

```

需要编码的类别列: ['class', 'employer_type', 'industry']
 编码完成, 数据形状: (10001, 21)

```

Out[35]:
   total_loan  year_of_loan  interest  monthly_payment  class  employer_type  indus
0  31818.18182           3.0    11.466           1174.91      2              3
1  28000.00000           5.0    16.841           670.69      2              3
2  17272.72727           3.0     8.900           603.32      0              3

```

3 rows × 21 columns

3.3 划分特征与标签

```

In [36]: X = df.drop(columns=['isDefault'])
y = df['isDefault']

```

```
print('特征矩阵形状:', X.shape)
print('标签形状:', y.shape)
print('特征列:', X.columns.tolist())
```

特征矩阵形状: (10001, 20)

标签形状: (10001,)

特征列: ['total_loan', 'year_of_loan', 'interest', 'monthly_payment', 'class', 'employer_type', 'industry', 'work_year', 'house_exist', 'use', 'debt_loan_ratio', 'delinquency_18month', 'known_outstanding_loan', 'known_dero', 'pub_dero_bankrup', 'early_return', 'early_return_amount', 'early_return_amount_3mon', 'issue_month', 'issue_year']

4. 数据集划分

```
In [37]: # 按 7:3 划分训练集和测试集, 固定随机种子保证可复现
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

print('训练集大小:', X_train.shape)
print('测试集大小:', X_test.shape)
print('训练集违约率: {:.2%}'.format(y_train.mean()))
print('测试集违约率: {:.2%}'.format(y_test.mean()))
```

训练集大小: (7000, 20)

测试集大小: (3001, 20)

训练集违约率: 16.83%

测试集违约率: 16.83%

```
In [38]: # 对逻辑回归和朴素贝叶斯使用标准化后的特征
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

5. 模型训练与评估

重点实现以下三个算法:

- 决策树 (Decision Tree)
- 逻辑回归 (Logistic Regression)
- 朴素贝叶斯 (Naive Bayes)

```
In [39]: # 定义一个评估函数, 方便复用
def evaluate_model(model_name, y_true, y_pred, y_prob):
    acc = accuracy_score(y_true, y_pred)
    prec = precision_score(y_true, y_pred, zero_division=0)
    rec = recall_score(y_true, y_pred, zero_division=0)
    f1 = f1_score(y_true, y_pred, zero_division=0)
    auc = roc_auc_score(y_true, y_prob)
    print(f'\n===== {model_name} =====')
    print(f'  准确率 Accuracy : {acc:.4f}')
    print(f'  精确率 Precision: {prec:.4f}')
    print(f'  召回率 Recall   : {rec:.4f}')
    print(f'  F1 Score       : {f1:.4f}')
```

```

print(f'   AUC           : {auc:.4f}')
print('\n分类报告: ')
print(classification_report(y_true, y_pred, target_names=['正常', '违约']))
return {'模型': model_name, 'Accuracy': acc, 'Precision': prec,
        'Recall': rec, 'F1': f1, 'AUC': auc}

```

5.1 决策树

```

In [40]: # 先用默认参数训练一棵决策树
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
dt_pred = dt.predict(X_test)
dt_prob = dt.predict_proba(X_test)[: , 1]

res_dt = evaluate_model('决策树 (默认参数)', y_test, dt_pred, dt_prob)

```

===== 决策树 (默认参数) =====

```

准确率 Accuracy : 0.8084
精确率 Precision: 0.4258
召回率 Recall   : 0.3980
F1 Score        : 0.4115
AUC              : 0.6447

```

分类报告:

	precision	recall	f1-score	support
正常	0.88	0.89	0.89	2496
违约	0.43	0.40	0.41	505
accuracy			0.81	3001
macro avg	0.65	0.64	0.65	3001
weighted avg	0.80	0.81	0.81	3001

```

In [41]: # 用 GridSearchCV 调参: max_depth 和 min_samples_split
dt_param_grid = {
    'max_depth': [3, 5, 8, 10, None],
    'min_samples_split': [2, 10, 20]
}
dt_gs = GridSearchCV(
    DecisionTreeClassifier(random_state=42),
    dt_param_grid, cv=5, scoring='f1', n_jobs=-1
)
dt_gs.fit(X_train, y_train)

print('决策树最优参数:', dt_gs.best_params_)
print('交叉验证最优 F1: {:.4f}'.format(dt_gs.best_score_))

dt_best = dt_gs.best_estimator_
dt_best_pred = dt_best.predict(X_test)
dt_best_prob = dt_best.predict_proba(X_test)[: , 1]

res_dt_best = evaluate_model('决策树 (调参后)', y_test, dt_best_pred, dt_best_prob)

```


决策树最优参数: {'max_depth': 3, 'min_samples_split': 2}
交叉验证最优 F1: 0.4568

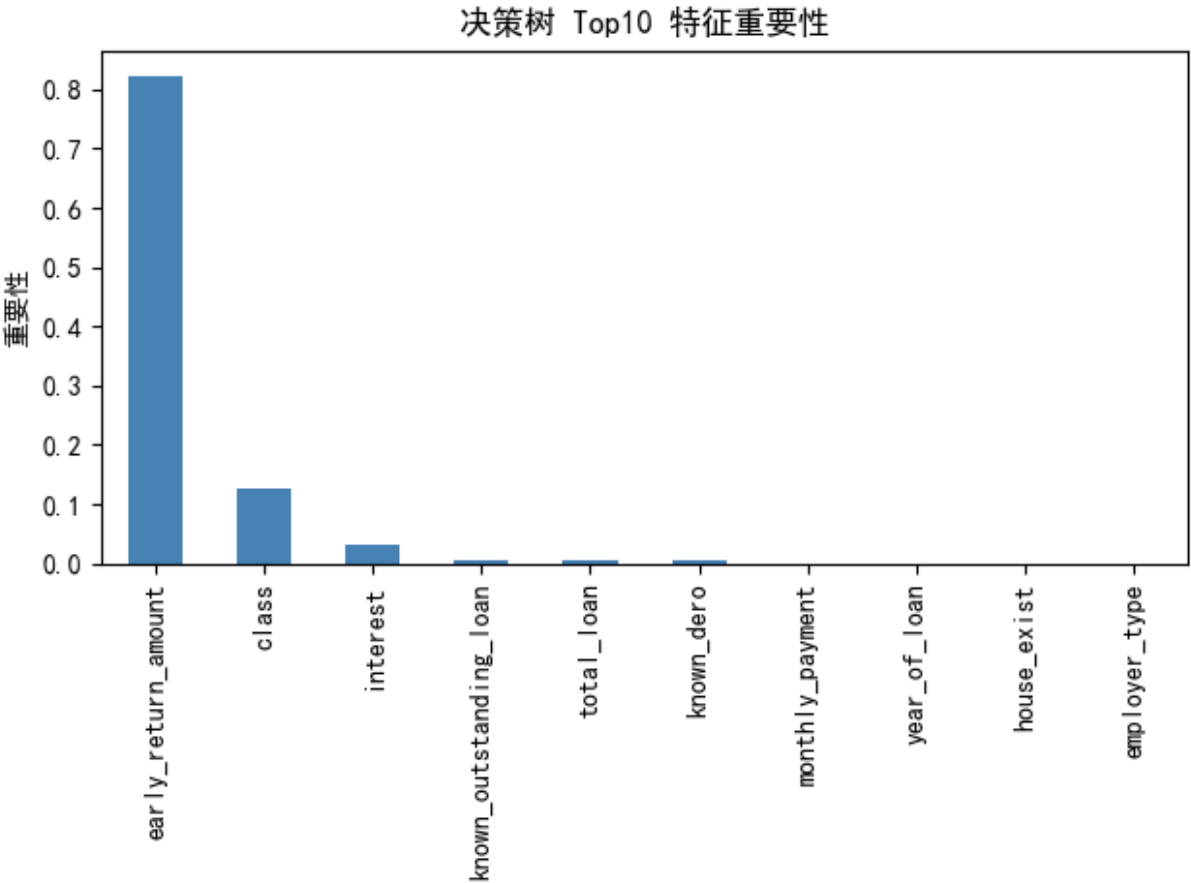
===== 决策树 (调参后) =====

准确率 Accuracy : 0.8417
精确率 Precision: 0.5408
召回率 Recall : 0.3941
F1 Score : 0.4559
AUC : 0.8567

分类报告:

	precision	recall	f1-score	support
正常	0.88	0.93	0.91	2496
违约	0.54	0.39	0.46	505
accuracy			0.84	3001
macro avg	0.71	0.66	0.68	3001
weighted avg	0.83	0.84	0.83	3001

```
In [42]: # 可视化决策树特征重要性
feat_imp = pd.Series(dt_best.feature_importances_, index=X.columns).sort_values(asc
feat_imp.head(10).plot(kind='bar', color='steelblue')
plt.title('决策树 Top10 特征重要性')
plt.ylabel('重要性')
plt.tight_layout()
plt.show()
```



5.2 逻辑回归

```
In [43]: # 默认参数逻辑回归
lr = LogisticRegression(max_iter=1000, random_state=42)
lr.fit(X_train_scaled, y_train)
lr_pred = lr.predict(X_test_scaled)
lr_prob = lr.predict_proba(X_test_scaled)[: , 1]

res_lr = evaluate_model('逻辑回归（默认参数）', y_test, lr_pred, lr_prob)
```

==== 逻辑回归（默认参数） =====

准确率 Accuracy : 0.8451
精确率 Precision: 0.5870
召回率 Recall : 0.2673
F1 Score : 0.3673
AUC : 0.8582

分类报告:

	precision	recall	f1-score	support
正常	0.87	0.96	0.91	2496
违约	0.59	0.27	0.37	505
accuracy			0.85	3001
macro avg	0.73	0.61	0.64	3001
weighted avg	0.82	0.85	0.82	3001

```
In [44]: # 调参: 正则化系数 C
lr_param_grid = {'C': [0.01, 0.1, 1, 10, 100]}
lr_gs = GridSearchCV(
    LogisticRegression(max_iter=1000, random_state=42),
    lr_param_grid, cv=5, scoring='f1', n_jobs=-1
)
lr_gs.fit(X_train_scaled, y_train)

print('逻辑回归最优参数: ', lr_gs.best_params_)
print('交叉验证最优 F1: {:.4f}'.format(lr_gs.best_score_))

lr_best = lr_gs.best_estimator_
lr_best_pred = lr_best.predict(X_test_scaled)
lr_best_prob = lr_best.predict_proba(X_test_scaled)[: , 1]

res_lr_best = evaluate_model('逻辑回归（调参后）', y_test, lr_best_pred, lr_best_prob)
```

逻辑回归最优参数: {'C': 1}
交叉验证最优 F1: 0.3455

===== 逻辑回归 (调参后) =====

准确率 Accuracy : 0.8451
精确率 Precision: 0.5870
召回率 Recall : 0.2673
F1 Score : 0.3673
AUC : 0.8582

分类报告:

	precision	recall	f1-score	support
正常	0.87	0.96	0.91	2496
违约	0.59	0.27	0.37	505
accuracy			0.85	3001
macro avg	0.73	0.61	0.64	3001
weighted avg	0.82	0.85	0.82	3001

5.3 朴素贝叶斯

```
In [45]: # 朴素贝叶斯 (高斯型, 适用于连续特征)
nb = GaussianNB()
nb.fit(X_train_scaled, y_train)
nb_pred = nb.predict(X_test_scaled)
nb_prob = nb.predict_proba(X_test_scaled)[: , 1]

res_nb = evaluate_model('朴素贝叶斯', y_test, nb_pred, nb_prob)
```

===== 朴素贝叶斯 =====

准确率 Accuracy : 0.7504
精确率 Precision: 0.3917
召回率 Recall : 0.8733
F1 Score : 0.5408
AUC : 0.8482

分类报告:

	precision	recall	f1-score	support
正常	0.97	0.73	0.83	2496
违约	0.39	0.87	0.54	505
accuracy			0.75	3001
macro avg	0.68	0.80	0.68	3001
weighted avg	0.87	0.75	0.78	3001

```
In [46]: # 调参: var_smoothing (控制数值稳定性的平滑参数)
nb_param_grid = {'var_smoothing': np.logspace(-12, 0, 13)}
nb_gs = GridSearchCV(
    GaussianNB(), nb_param_grid, cv=5, scoring='f1', n_jobs=-1
)
nb_gs.fit(X_train_scaled, y_train)
```

```

print('朴素贝叶斯最优参数: ', nb_gs.best_params_)
print('交叉验证最优 F1: {:.4f}'.format(nb_gs.best_score_))

nb_best = nb_gs.best_estimator_
nb_best_pred = nb_best.predict(X_test_scaled)
nb_best_prob = nb_best.predict_proba(X_test_scaled)[: , 1]

res_nb_best = evaluate_model('朴素贝叶斯 (调参后)', y_test, nb_best_pred, nb_best_pr

```

朴素贝叶斯最优参数: {'var_smoothing': np.float64(0.001)}
交叉验证最优 F1: 0.5310

===== 朴素贝叶斯 (调参后) =====

准确率 Accuracy : 0.7514
精确率 Precision: 0.3925
召回率 Recall : 0.8713
F1 Score : 0.5412
AUC : 0.8481

分类报告:

	precision	recall	f1-score	support
正常	0.97	0.73	0.83	2496
违约	0.39	0.87	0.54	505
accuracy			0.75	3001
macro avg	0.68	0.80	0.69	3001
weighted avg	0.87	0.75	0.78	3001

6. 模型效果比较

In [47]: # 汇总所有模型 (调参后) 的指标

```

results = pd.DataFrame([
    res_dt_best,
    res_lr_best,
    res_nb_best
])
results.set_index('模型', inplace=True)
print(results.round(4))

```

	Accuracy	Precision	Recall	F1	AUC
模型					
决策树 (调参后)	0.8417	0.5408	0.3941	0.4559	0.8567
逻辑回归 (调参后)	0.8451	0.5870	0.2673	0.3673	0.8582
朴素贝叶斯 (调参后)	0.7514	0.3925	0.8713	0.5412	0.8481

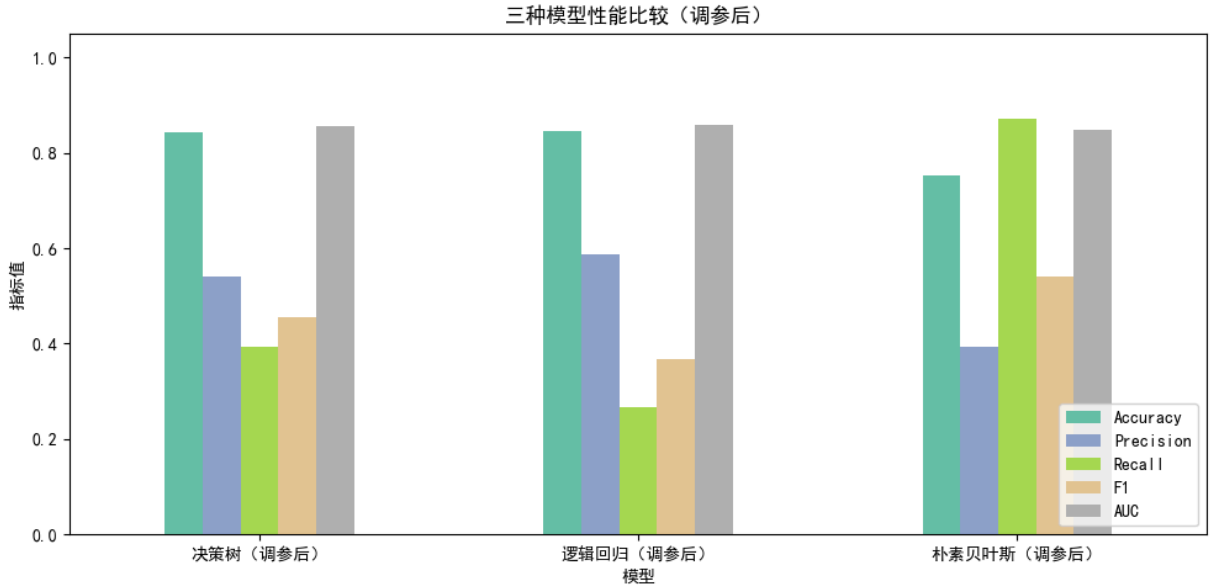
In [48]: # 柱状图比较各模型主要指标

```

results[['Accuracy', 'Precision', 'Recall', 'F1', 'AUC']].plot(
    kind='bar', figsize=(10, 5), colormap='Set2'
)
plt.title('三种模型性能比较 (调参后)')
plt.ylabel('指标值')
plt.ylim(0, 1.05)

```

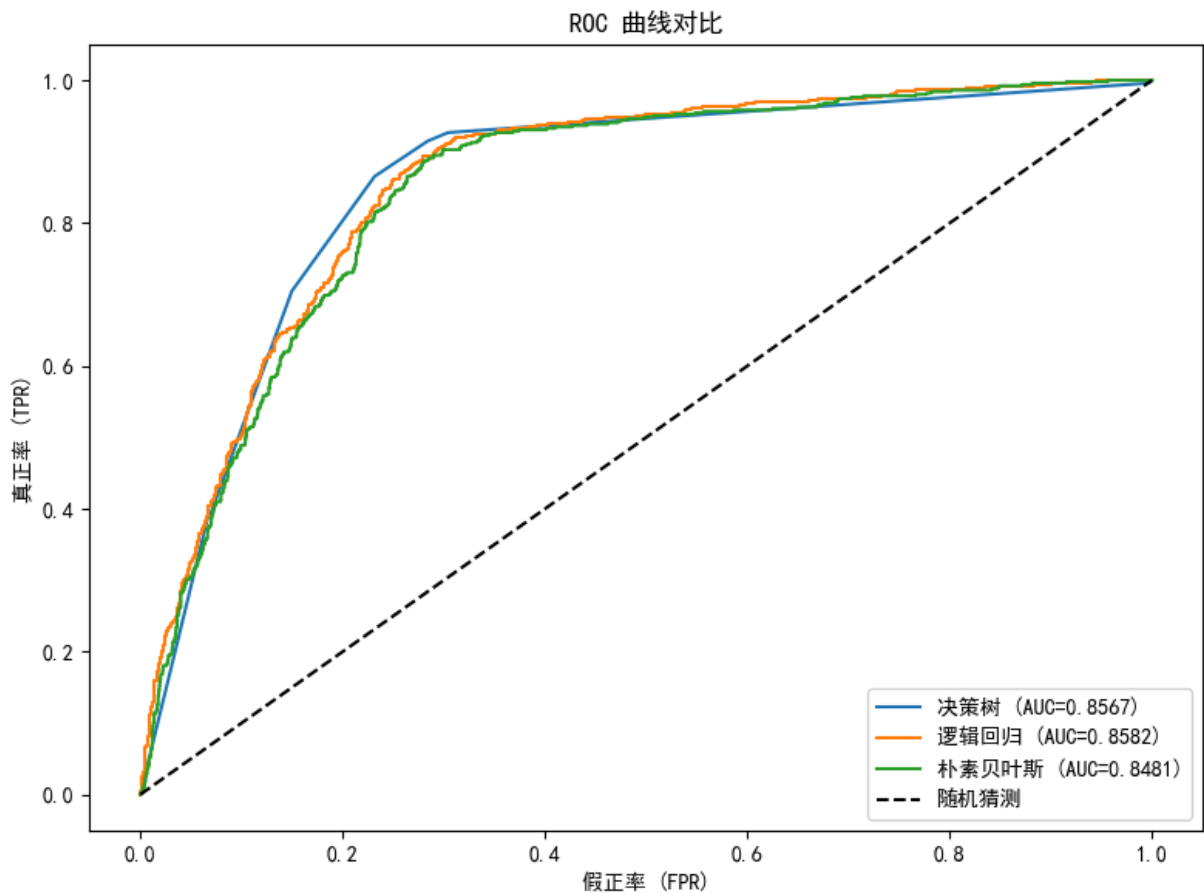
```
plt.xticks(rotation=0)
plt.legend(loc='lower right')
plt.tight_layout()
plt.show()
```



```
In [49]: # ROC 曲线对比
plt.figure(figsize=(8, 6))

for name, prob in [
    ('决策树', dt_best_prob),
    ('逻辑回归', lr_best_prob),
    ('朴素贝叶斯', nb_best_prob)
]:
    fpr, tpr, _ = roc_curve(y_test, prob)
    auc_val = roc_auc_score(y_test, prob)
    plt.plot(fpr, tpr, label=f'{name} (AUC={auc_val:.4f})')

plt.plot([0, 1], [0, 1], 'k--', 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()
plt.tight_layout()
plt.show()
```

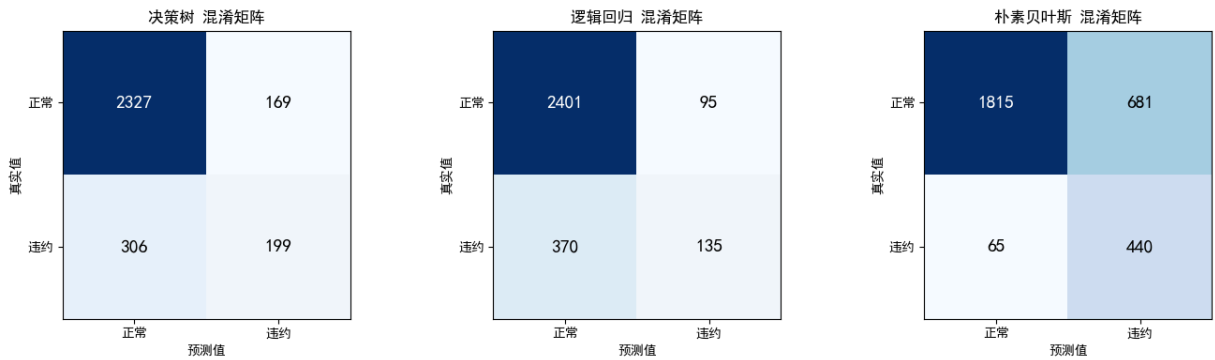


```
In [50]: # 混淆矩阵
fig, axes = plt.subplots(1, 3, figsize=(14, 4))

model_results = [
    ('决策树', dt_best_pred),
    ('逻辑回归', lr_best_pred),
    ('朴素贝叶斯', nb_best_pred)
]

for ax, (name, pred) in zip(axes, model_results):
    cm = confusion_matrix(y_test, pred)
    im = ax.imshow(cm, cmap='Blues')
    ax.set_title(f'{name} 混淆矩阵')
    ax.set_xlabel('预测值')
    ax.set_ylabel('真实值')
    ax.set_xticks([0, 1])
    ax.set_yticks([0, 1])
    ax.set_xticklabels(['正常', '违约'])
    ax.set_yticklabels(['正常', '违约'])
    for i in range(2):
        for j in range(2):
            ax.text(j, i, cm[i, j], ha='center', va='center',
                    color='white' if cm[i, j] > cm.max() / 2 else 'black', fontsize=12)

plt.tight_layout()
plt.show()
```



7. 总结

本实验使用包含约 10000 条个人贷款记录的数据集（**违约率约为 16.83%**），对**决策树、逻辑回归、朴素贝叶斯**三种算法进行了训练与调参，测试集上的运行结果如下：

7.1 各模型指标对比（调参后，测试集）

模型	最优参数	Accuracy	Precision	Recall	F1	AUC
决策树	max_depth=3, min_samples_split=2	0.8417	0.5408	0.3941	0.4559	0.8567
逻辑回归	C=1	0.8451	0.5870	0.2673	0.3673	0.8582
朴素贝叶斯	var_smoothing=0.001	0.7514	0.3925	0.8713	0.5412	0.8481

7.2 结果分析

AUC 排名：逻辑回归 (0.8582) > 决策树 (0.8567) > 朴素贝叶斯 (0.8481)，三者相差不超过 0.01，说明在区分违约与正常用户的整体能力上三种模型表现接近。

- **逻辑回归**准确率最高（84.51%），精确率最好（0.5870），但召回率仅 0.2673，也就是说真实违约用户中只有约 27% 被识别出来，容易漏报风险，F1 仅 0.3673，是三者最低的。
- **决策树**调参后最优深度仅为 3，说明用浅层规则就能抓住主要违约信号，精确率（0.5408）和召回率（0.3941）相对均衡，F1 为 0.4559，整体介于另外两者之间。
- **朴素贝叶斯**准确率最低（75.14%），但召回率高达 0.8713，能识别出约 87% 的真实违约用户，F1 达到 0.5412，是三者中最高的。在信贷风控场景下，**漏掉违约用户的损失远大于误判正常用户**，从这个角度看朴素贝叶斯最符合业务需求。

7.3 关键特征（决策树特征重要性 Top 5）

排名	特征	重要性得分
1	early_return_amount（提前还款累积金额）	0.8222

排名	特征	重要性得分
2	class (贷款级别)	0.1266
3	interest (贷款利率)	0.0323
4	known_outstanding_loan (未结信用额度数量)	0.0069
5	total_loan (贷款金额)	0.0063

提前还款金额 (early_return_amount) 的重要性高达 0.82，独占主导地位，远超其他所有特征——有过大额提前还款记录的借款人资金充裕、还款意愿强，违约概率极低；反之几乎没有提前还款记录的用户风险显著偏高。贷款级别 (class) 和利率 (interest) 分别排第 2、3 位，说明银行在放款时的风险定价本身已经隐含了部分违约信息。

7.4 综合建议

- 若业务目标是**尽量不漏掉违约用户**（控制坏账损失），优先选用**朴素贝叶斯** (Recall=0.87) ；
- 若要求**预测结果更精准、误报更少**（减少对优质客户的误拒），选用**逻辑回归** (Precision=0.59) ；
- **决策树**可解释性最强（深度仅 3 层），适合向业务方展示规则、辅助信贷审批决策。