1. Markov Model

import numpy as np

import matplotlib.pyplot as plt

import networkx as nx

import seaborn as sns

# 设置中文字体

plt.rcParams['font.sans-serif'] = ['SimHei']

plt.rcParams['axes.unicode\_minus'] = False

# 基于行业基准的状态转移矩阵

P = np.array([[0.90, 0.10],  # 续订->续订: 90%, 续订->流失: 10%

              [0.00, 1.00]]) # 流失->续订: 0%,  流失->流失: 100%

# 不同套餐的基准留存率（考虑价格和用户粘性）

plan\_retention = {

    'Basic': 0.85,    # 基础套餐留存率较低

    'Standard': 0.90, # 标准套餐符合行业平均

    'Premium': 0.93   # 高端套餐留存率较高

}

# 模拟参数

n\_users = 10000  # 样本量

n\_months = 24    # 观察期

initial\_state = np.array([1, 0])  # 初始全部为续订用户

# 生成用户分布

plans = np.random.choice(['Basic', 'Standard', 'Premium'],

                        size=n\_users,

                        p=[0.3, 0.5, 0.2])  # 套餐分布

# 生成用户留存矩阵

retention\_matrix = np.zeros((n\_users, n\_months))

retention\_matrix[:, 0] = 1  # 初始状态

# 模拟用户行为

for month in range(1, n\_months):

    for user in range(n\_users):

        if retention\_matrix[user, month-1] == 1:

            retention\_prob = plan\_retention[plans[user]]

            retention\_matrix[user, month] = np.random.binomial(1, retention\_prob)

# 计算各项指标

monthly\_retention\_rates = []

for month in range(1, n\_months):

    active\_prev = np.sum(retention\_matrix[:, month-1])

    active\_curr = np.sum(retention\_matrix[:, month])

    rate = active\_curr / active\_prev if active\_prev > 0 else 0

    monthly\_retention\_rates.append(rate)

# 按套餐类型分析留存率

plan\_retention\_rates = {}

for plan in plan\_retention.keys():

    plan\_users = plans == plan

    plan\_matrix = retention\_matrix[plan\_users]

    rates = []

    for month in range(1, n\_months):

        active\_prev = np.sum(plan\_matrix[:, month-1])

        active\_curr = np.sum(plan\_matrix[:, month])

        rate = active\_curr / active\_prev if active\_prev > 0 else 0

        rates.append(rate)

    plan\_retention\_rates[plan] = rates

# 可视化分析

plt.figure(figsize=(15, 5))

# 1. 整体留存率趋势

plt.subplot(131)

plt.plot(range(1, n\_months), monthly\_retention\_rates, 'b-', label='Actual Retention')

plt.axhline(y=0.90, color='r', linestyle='--', label='Industry Benchmark')

plt.title('Monthly Retention Rate Trend')

plt.xlabel('Month')

plt.ylabel('Retention Rate')

plt.legend()

plt.grid(True)

# 2. 不同套餐留存率对比

plt.subplot(132)

for plan, rates in plan\_retention\_rates.items():

    plt.plot(range(1, n\_months), rates, label=plan)

plt.title('Retention Rate by Plan Type')

plt.xlabel('Month')

plt.ylabel('Retention Rate')

plt.legend()

plt.grid(True)

# 3. 用户分布变化

plt.subplot(133)

active\_users = np.sum(retention\_matrix, axis=0) / n\_users

plt.plot(range(n\_months), active\_users, 'g-')

plt.title('Active Users Ratio Over Time')

plt.xlabel('Month')

plt.ylabel('Active Users Ratio')

plt.grid(True)

plt.tight\_layout()

plt.show()

# 用户流失预测图

plt.figure(figsize=(12, 5))

time = np.arange(n\_months + 1)  # 使用 n\_months 替代 n\_predict

plt.plot(time[:-1], active\_users, 'b-o', label="Active Users", linewidth=2)

plt.plot(time[:-1], 1 - active\_users, 'r--o', label="Churned Users", linewidth=2)

plt.title("Netflix User Churn Prediction (Industry Benchmark)", fontsize=14)

plt.xlabel("Month", fontsize=12)

plt.ylabel("User Ratio", fontsize=12)

plt.xticks(time[:-1])

plt.legend(fontsize=10)

plt.grid(True, linestyle='--', alpha=0.7)

plt.show()

# 优化马尔可夫链状态转移图

plt.figure(figsize=(20, 10))

G = nx.DiGraph()

# 定义多层级节点

levels = {

    0: [("Active", "green")],

    1: [("Retained", "green"), ("Churned\_Price", "gray")],

    2: [("Loyal", "green"), ("At\_Risk", "green"), ("Lost", "gray"), ("Returned", "green")],

    3: [("Stable", "green"), ("Shared", "gray"), ("Competitor", "gray"), ("Content", "gray")]

}

# 定义转移概率

transitions = [

    # 第一层到第二层

    ("Active", "Retained", {"weight": 0.90}),

    ("Active", "Churned\_Price", {"weight": 0.10}),

    # 第二层到第三层

    ("Retained", "Loyal", {"weight": 0.70}),

    ("Retained", "At\_Risk", {"weight": 0.30}),

    ("Churned\_Price", "Lost", {"weight": 0.50}),

    ("Churned\_Price", "Returned", {"weight": 0.50}),

    # 第三层到第四层

    ("Loyal", "Stable", {"weight": 0.95}),

    ("At\_Risk", "Shared", {"weight": 0.40}),

    ("At\_Risk", "Competitor", {"weight": 0.60}),

    ("Lost", "Content", {"weight": 1.00})

]

# 添加节点和边

for level in levels.values():

    for node, color in level:

        G.add\_node(node, color=color)

G.add\_edges\_from(transitions)

# 设置层级布局

pos = {}

for level\_idx, level\_nodes in levels.items():

    n\_nodes = len(level\_nodes)

    for i, (node, \_) in enumerate(level\_nodes):

        pos[node] = np.array([level\_idx \* 4, (i - (n\_nodes-1)/2) \* 2])

# 绘制节点

for node, (x, y) in pos.items():

    color = G.nodes[node]['color']

    nx.draw\_networkx\_nodes(G, {node: (x, y)},

                          nodelist=[node],

                          node\_color=color,

                          node\_size=2500,

                          alpha=0.7)

# 绘制节点标签

nx.draw\_networkx\_labels(G, pos, font\_size=10, font\_weight='bold')

# 绘制边和箭头

for edge in G.edges(data=True):

    nx.draw\_networkx\_edges(G, pos,

                          edgelist=[(edge[0], edge[1])],

                          edge\_color='gray',

                          arrows=True,

                          arrowsize=20,

                          width=1.5)

# 添加转移概率标签

edge\_labels = {(u, v): f"{d['weight']:.2f}" for u, v, d in G.edges(data=True)}

nx.draw\_networkx\_edge\_labels(G, pos,

                           edge\_labels=edge\_labels,

                           font\_size=9)

plt.title("Netflix User State Transition Tree", fontsize=16, pad=20)

plt.axis('off')

plt.tight\_layout()

1. Logisitic Regression Model

导入库

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.metrics import accuracy\_score, roc\_auc\_score, recall\_score, confusion\_matrix, roc\_curve

# 设置随机种子

np.random.seed(42)

# 数据生成

n = 10000

watch\_hours\_weekly = np.random.normal(loc=9, scale=4, size=n).clip(0, 30)

plan\_type = np.random.choice(['Basic', 'Standard', 'Premium'], size=n, p=[0.3, 0.5, 0.2])

use\_mobile\_only = np.random.choice([True, False], size=n, p=[0.4, 0.6])

shared\_account = np.random.choice([True, False], size=n, p=[0.5, 0.5])

watched\_netflix\_original = np.random.choice([True, False], size=n, p=[0.7, 0.3])

paused\_account = np.random.poisson(lam=0.3, size=n)

# 流失概率构建

churn\_prob = (

    0.3 \* (watch\_hours\_weekly < 5).astype(int) +

    0.2 \* use\_mobile\_only.astype(int) +

    0.2 \* (plan\_type == 'Basic').astype(int) +

    0.15 \* (~watched\_netflix\_original).astype(int) +

    0.1 \* (paused\_account >= 1).astype(int) +

    0.05 \* shared\_account.astype(int)

)

churn\_prob += np.random.normal(0, 0.05, size=n)

churn\_prob = np.clip(churn\_prob, 0, 1)

churned = np.random.binomial(1, churn\_prob)

# 构建DataFrame

df = pd.DataFrame({

    'watch\_hours\_weekly': watch\_hours\_weekly,

    'plan\_type': plan\_type,

    'use\_mobile\_only': use\_mobile\_only,

    'shared\_account': shared\_account,

    'watched\_netflix\_original': watched\_netflix\_original,

    'paused\_account': paused\_account,

    'churned': churned

})

# 保存为CSV

df.to\_csv("netflix\_churn\_simulated.csv", index=False)

# 基础可视化

plt.figure(figsize=(8, 5))

sns.boxplot(x='churned', y='watch\_hours\_weekly', data=df)

plt.title('每周观看时长与流失关系')

plt.xlabel('是否流失')

plt.ylabel('每周平均观看小时')

plt.show()

plt.figure(figsize=(6, 4))

sns.barplot(x='plan\_type', y='churned', data=df)

plt.title('不同套餐的平均流失率')

plt.xlabel('套餐类型')

plt.ylabel('流失率')

plt.show()

# 数据预处理

df\_encoded = pd.get\_dummies(df, columns=['plan\_type'], drop\_first=True)

df\_encoded[['use\_mobile\_only', 'shared\_account', 'watched\_netflix\_original']] = df\_encoded[

    ['use\_mobile\_only', 'shared\_account', 'watched\_netflix\_original']].astype(int)

# 特征与标签分离

X = df\_encoded.drop('churned', axis=1)

y = df\_encoded['churned']

# 训练集/测试集划分

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# 模型训练

model = LogisticRegression(max\_iter=1000)

model.fit(X\_train, y\_train)

# 模型评估

y\_pred = model.predict(X\_test)

y\_proba = model.predict\_proba(X\_test)[:, 1]

print("准确率 Accuracy: ", accuracy\_score(y\_test, y\_pred))

print("召回率 Recall: ", recall\_score(y\_test, y\_pred))

print("AUC值: ", roc\_auc\_score(y\_test, y\_proba))

# 混淆矩阵

cm = confusion\_matrix(y\_test, y\_pred)

plt.figure()

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.title('混淆矩阵')

plt.xlabel('预测值')

plt.ylabel('实际值')

plt.show()

# ROC曲线

fpr, tpr, \_ = roc\_curve(y\_test, y\_proba)

plt.figure(figsize=(6, 4))

plt.plot(fpr, tpr, label=f"AUC = {roc\_auc\_score(y\_test, y\_proba):.2f}")

plt.plot([0, 1], [0, 1], linestyle='--')

plt.xlabel('假阳性率 (FPR)')

plt.ylabel('真阳性率 (TPR)')

plt.title('ROC 曲线')

plt.legend()

plt.show()

# 模型系数输出

coef\_df = pd.DataFrame({

    'Feature': X.columns,

    'Coefficient': model.coef\_[0]

}).sort\_values(by='Coefficient', ascending=False)

print(coef\_df)

# 添加交叉验证

cv\_scores = cross\_val\_score(model, X\_train, y\_train, cv=5)

print(f"交叉验证分数: {cv\_scores.mean():.3f} (+/- {cv\_scores.std() \* 2:.3f})")

# 特征相关性热力图

plt.figure(figsize=(10, 8))

correlation = df\_encoded.corr()

sns.heatmap(correlation, annot=True, cmap='coolwarm', center=0)

plt.title('特征相关性热力图')

plt.show()

# 分类特征流失率对比

categorical\_features = ['use\_mobile\_only', 'shared\_account', 'watched\_netflix\_original']

fig, axes = plt.subplots(1, 3, figsize=(15, 4))

for i, feature in enumerate(categorical\_features):

    sns.barplot(x=feature, y='churned', data=df, ax=axes[i])

    axes[i].set\_title(f'{feature}与流失率关系')

plt.tight\_layout()

plt.show()

# 特征重要性可视化

plt.figure(figsize=(10, 6))

sns.barplot(x='Coefficient', y='Feature', data=coef\_df)

plt.title('特征重要性')

plt.show()

# 阈值分析

thresholds = np.arange(0.1, 1.0, 0.1)

scores = []

for threshold in thresholds:

    y\_pred\_threshold = (y\_proba >= threshold).astype(int)

    scores.append({

        'threshold': threshold,

        'accuracy': accuracy\_score(y\_test, y\_pred\_threshold),

        'recall': recall\_score(y\_test, y\_pred\_threshold)

    })

threshold\_df = pd.DataFrame(scores)

plt.figure(figsize=(8, 5))

threshold\_df.plot(x='threshold', y=['accuracy', 'recall'])

plt.title('不同阈值下的模型表现')

plt.show()