实验一：通信客户流失预警模型

实验报告

刘东洋 1120212831

1 挖掘过程

1.1 业务理解

使用分类模型构建客户流失预测模型，通过客户为流失客户的概率预测该客户是否为流失客户并根据客户为流失客户的概率生成流失概率排序名单。

1.2 数据理解

在本案例中使用的数据来自于 IBM Sample Data Sets，是某电信公司一段时间内的客户消费数据。共包含 7043 笔客户资料，每笔客户资料包含 21 个字段，其中 1 个客户 ID 字段，19 个属性特征字段及 1 个标签字段（Yes 代表流失，No 代表未流失）。属性特征字段主要包含以下三个维度指标：客户画像指标（如性别、是否老年等）、消费产品指标（如是否开通互联网服务、是否开通电话服务等）、消费信息指标（如付款方式、月费用等）。字段的具体说明如表 1 所示：

字段	字段翻译	角色	特征类型	不同值个数	备注
customerID	客户 ID	ID	无类型	7043	
Gender	性别	特征	类别	2	
SeniorCitizen	是否为老年人	特征	类别	2	初始取值为 0/1，分别代表否/是
Partener	是否有配偶	特征	类别	2	
Dependents	是否和需抚养/赡养人同居	特征	类别	2	
Tenure	在网时长	特征	数值		单位：月
PhoneService	是否开通电话服务业务	特征	类别	2	
MultipleLines	是否开通多线业务	特征	类别	3	取值为是/否/未开通电话业务
InternetService	是否开通互联网服务	特征	类别	3	取值为否/DSL/光纤
OnlineSecurity	是否开通网络安全服务	特征	类别	3	取值为是/否/未开通互联网服务
OnlineBackup	是否开通在线备份业	特征	类别	3	取值为是/否/未

	务				开通互联网服务
DeviceProtection	是否开通了设备保护业务	特征	类别	3	取值为是/否/未开通互联网服务
TechSupport	是否开通了技术支持服务	特征	类别	3	取值为是/否/未开通互联网服务
StreamingTV	是否开通网络电视	特征	类别	3	取值为是/否/未开通互联网服务
StreamingMovies	是否开通网络电影	特征	类别	3	取值为是/否/未开通互联网服务
Contract	签订合同方式	特征	类别	3	
PaperlessBillling	是否开通电子账单	特征	类别	2	
PaymentMethod	付款方式	特征	类别	4	
MonthlyCharges	月费用	特征	数值		单位：美元
TotalCharges	总费用	特征	数值		单位：美元
Churn	是否流失	标签类别	类别	2	取值为是/否

表 1 数据集字段说明

1.3 数据准备

本案例数据准备工作步骤如下。

（1）读入数据集

代码如表 2 所示：

读入数据集代码

```
import numpy as np
import pandas as pd

df = pd.read_csv('./Telco-Customer-Churn.csv')
df.head()
```

表 2 读入数据集代码

（2）数据初步清洗

首先进行初步的数据清洗工作，包含缺失值、错误值和异常值处理，并识别字段类型。同时通过箱线图对数值型特征检查异常点,其中清洗工作主要包含：

a) 对 MultipleLines（是否开通多线服务）、OnlineSecurity（是否开通网络安全服务）、OnlineBackup（是否开通在线备份业务）、DeviceProtection（是否开通了设备保护业务）、TechSupport（是否开通了技术支持服务）、StreamingTV（是否开通网络电视）、StreamingMovies（是否开通网络电影）等属性特征进行错误值处理。如不满足前置条件，则这些特征直接取值为 No（若未开通互联网服务，自然不会开通网络电视等服务）。

b) 对 TotalCharges（总费用）特征进行异常值处理。将空白取值替换空值 nan，由于含空

值的行较少，将其一并删除。删除后该特征取值只有数值，因此将此特征类型设为浮点型。

c) 对 Tenure（在网时长）属性离散化。Tenure 取值为[0, 72]，为适应后续决策树建模，将其离散化，以 12、24、36、48、60 为临界值进行分箱操作，共分为 6 组。

d) 判断字段的类别

e) 检查是否有缺失值很高(超过 50% 或 70%)的特征,需要删除。

f) 对数值型特征检查异常点,如通过箱线图或离群点检测等方法。

数据清洗代码如表 3 所示：

数据初步清洗代码

```
# 检查特征缺失值情况
null_cols = df.isnull().sum()[df.isnull().sum() > 0]
print(null_cols[null_cols / len(df) > 0.4])

# 数值特征异常点检查
import matplotlib.pyplot as plt

cat_cols = df.select_dtypes(include=['object']).columns
num_cols = df.select_dtypes(include=['float', 'int']).columns

for col in num_cols:
    fig = plt.figure(figsize=(8, 4))
    plt.boxplot(df[col])
    plt.grid(True)
    plt.title(col)
    plt.show()

# 错误值处理
repl_columns = ['OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
                'TechSupport','StreamingTV', 'StreamingMovies']

for i in repl_columns:
    df[i]  = df[i].replace({'No internet service' : 'No'})

# 替换值 SeniorCitizen，便于后续探索性分析
df["SeniorCitizen"] = df["SeniorCitizen"].replace({1: "Yes", 0: "No"})

# 替换值 TotalCharges
df['TotalCharges'] = df['TotalCharges'].replace(' ', np.nan)

# TotalCharges 空值：数据量小，直接删除
df = df.dropna(subset=['TotalCharges'])
df.reset_index(drop=True, inplace=True)  # 重置索引

# 转换数据类型
```

```
df['TotalCharges'] = df['TotalCharges'].astype('float')

# 转换 tenure
def transform_tenure(x):
    if x <= 12:
        return 'Tenure_1'
    elif x <= 24:
        return 'Tenure_2'
    elif x <= 36:
        return 'Tenure_3'
    elif x <= 48:
        return 'Tenure_4'
    elif x <= 60:
        return 'Tenure_5'
    else:
        return('Tenure_over_5')

df['tenure_group'] = df.tenure.apply(transform_tenure)

# 数值型和类别型字段
Id_col = ['customerID']
target_col = ['Churn']

cat_cols = df.nunique()[df.nunique() < 10].index.tolist()
num_cols = [i for i in df.columns if i not in cat_cols + Id_col]

print('类别型字段：\n', cat_cols)
print('-' * 30)
print('数值型字段：\n', num_cols)
```

表 3 数据初步清洗代码

输出结果如图 1 所示：

```
类别型字段:
 ['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines',
'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod', 'Chu
rn', 'tenure_group']
------------------------------
数值型字段:
 ['tenure', 'MonthlyCharges', 'TotalCharges']
```

图 1 类别型字段与数值型字段

特征缺失值情况检查结果：

```
Series([], dtype: int64)
```

图 2 特征缺失值情况检查结果：

Series([], dtype: int64)意味着:

df.isnull().sum()计算每个特征的缺失值数,没有特征缺失值超过样本长度的 40%。
因此 null_cols 为空 Series,打印时输出[]
所以这说明:
数据整体质量好,没有哪个特征缺失值比例过高
程序运行正确,没有报异常

数值特征异常点检查:

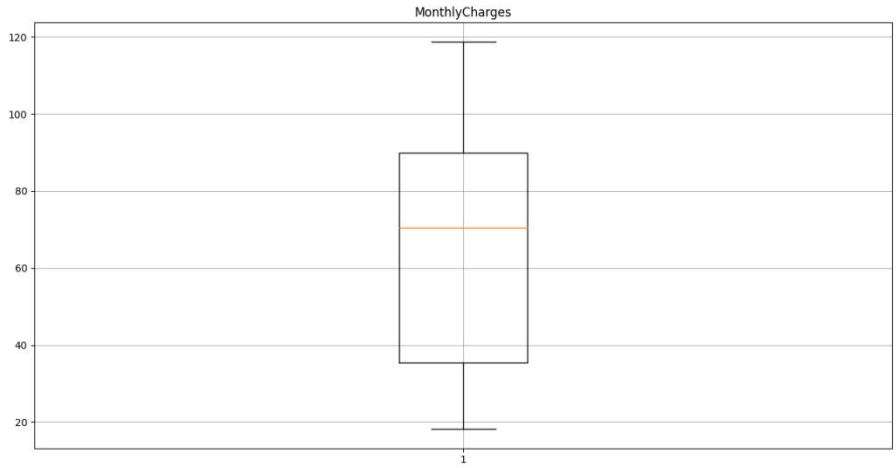


图 3 MonthlyCharges 箱线图：

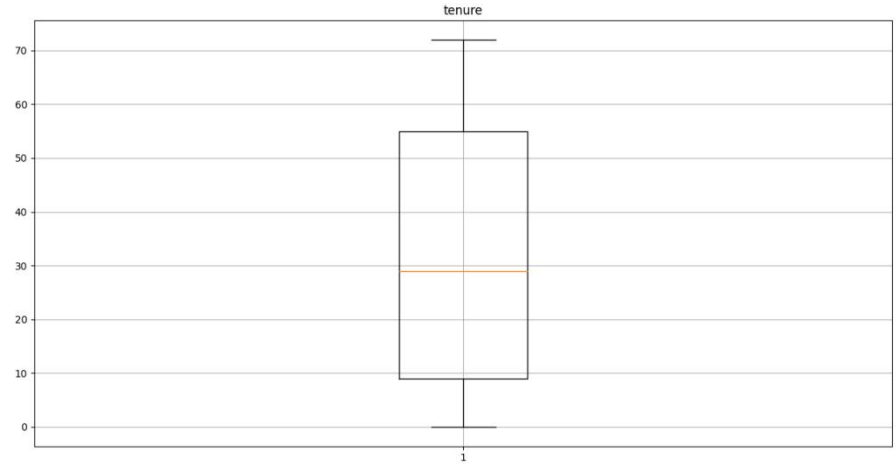


图 4 Tenure 箱线图

两个特征的箱线图都没有观察到明显的离群点。所有数据点主要集中在箱体内,没有异常值点较远从整体分布偏离。

这表明两个特征在原始数据中的分布比较紧凑,数据质量较好没有明显异常问题。

（3）探索性分析

对指标进行归纳梳理，分用户画像指标，消费产品指标，消费信息指标。探索影响用户

流失的关键因素。

a) 首先查看流失用户与非流失用户的整体分布情况。

用户整体分布情况代码如表 3 所示：

用户整体分布情况代码

```
# 探索性分析
import matplotlib.pyplot as plt
import seaborn as sns
import plotly as py
import plotly.graph_objs as go
import plotly.figure_factory as ff

# 目标变量 Churn 分布
df['Churn'].value_counts()

trace0 = go.Pie(labels=['未流失客户', '流失客户'],
                #                labels=df['Churn'].value_counts().index,
                values=df['Churn'].value_counts().values,
                hole=.5,
                rotation=90,
                marker=dict(colors=['rgb(154,203,228)', 'rgb(191,76,81)'],
                            line=dict(color='white', width=1.3))
                )
data = [trace0]
layout = go.Layout(title='目标变量 Churn 分布', font=dict(size=26))

fig = go.Figure(data=data, layout=layout)
py.offline.plot(fig, filename='整体流失情况分布.html', auto_open=False)
# 探索性分析
import matplotlib.pyplot as plt
import seaborn as sns
import plotly as py
import plotly.graph_objs as go
import plotly.figure_factory as ff

# 目标变量 Churn 分布
df['Churn'].value_counts()

trace0 = go.Pie(labels=['未流失客户', '流失客户'],
                #                labels=df['Churn'].value_counts().index,
                values=df['Churn'].value_counts().values,
                hole=.5,
                rotation=90,
                marker=dict(colors=['rgb(154,203,228)', 'rgb(191,76,81)'],
                            line=dict(color='white', width=1.3))
```

```
                    )
data = [trace0]
layout = go.Layout(title='目标变量 Churn 分布', font=dict(size=26))

fig = go.Figure(data=data, layout=layout)
py.offline.plot(fig, filename='整体流失情况分布.html', auto_open=False)
# 探索性分析
```

表 4 整体流失情况分布代码

输出结果如图所示：

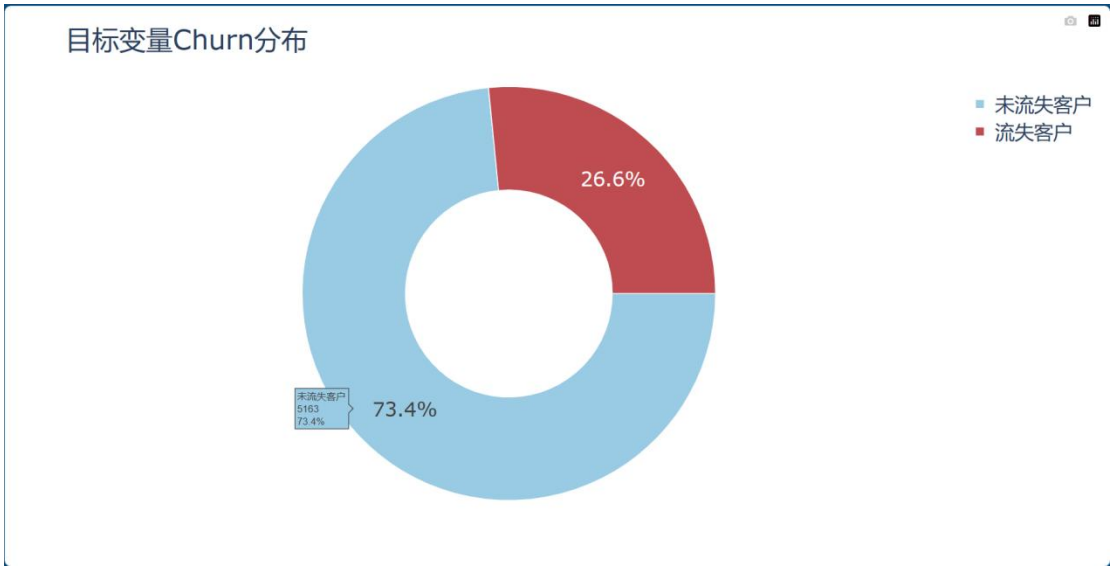


图 5 整体流失情况

结果：经过初步清洗之后的数据集大小为 7032 条记录，其中流失客户为 1869 条，占比 26.6%，未流失客户占比 73.4%。(略微不平衡)

再分别探索对各特征的影响。将每一个变量不同取值条件下用户流失与否用柱形图表示出来，可以直观地看出变量取值是否影响用户流失。

b) 查看类别型变量取值与是否流失的关系。

类别型变量与流失关系代码如表 5 所示：

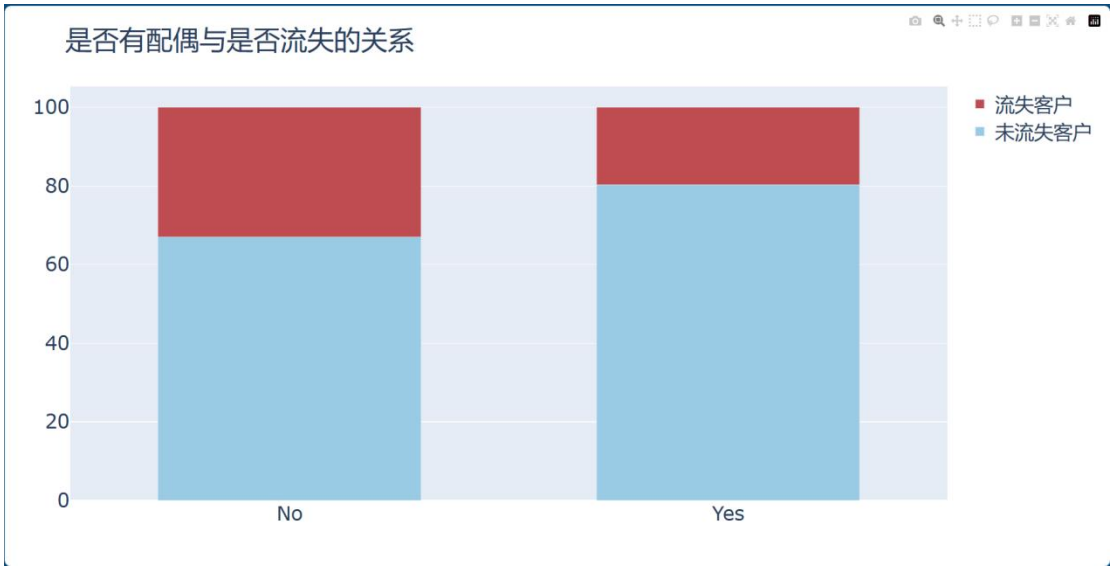
类别型变量取值与是否流失的关系代码

```
def plot_bar(input_col: str, target_col: str, title_name: str):
    cross_table = round(pd.crosstab(df[input_col], df[target_col], normalize='index') * 100, 2)

    # 索引
    index_0 = cross_table.columns.tolist()[0]
    index_1 = cross_table.columns.tolist()[1]

    # 绘图轨迹
    trace0 = go.Bar(x=cross_table.index.tolist(),
                    y=cross_table[index_0].values.tolist(),
                    #                    name=index_0,
                    marker=dict(color='rgb(154,203,228)'),
                    name='未流失客户'
```

```
                    )
    trace1 = go.Bar(x=cross_table.index.tolist(),
                    y=cross_table[index_1].values.tolist(),
                    #                     name=index_1,
                    marker=dict(color='rgb(191,76,81)'),
                    name='流失客户'
                    )

    data = [trace0, trace1]
    # 布局
    layout = go.Layout(title=title_name, bargap=0.4, barmode='stack', font=dict(size=26))

    # 画布
    fig = go.Figure(data=data, layout=layout)
    # 绘图
    py.offline.plot(fig, filename=f'./html/{title_name}.html', auto_open=False)


# 性别与是否流失的关系
chars = ['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines', 'InternetService',
         'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
         'Contract', 'PaperlessBilling', 'PaymentMethod', 'Churn', 'tenure_group']
# plot_bar(input_col='tenure_group', target_col='Churn', title_name='在网时长与是否流失的关系')
# plot_bar(input_col='gender', target_col='Churn', title_name='性别与是否流失的关系')
# plot_bar(input_col='SeniorCitizen', target_col='Churn', title_name='是否为老年人与是否流失的关系')
# plot_bar(input_col='Dependents', target_col='Churn', title_name='是否独立与是否流失的关系')
plot_bar(input_col='Partner', target_col='Churn', title_name='是否有配偶与是否流失的关系')
plot_bar(input_col='PhoneService', target_col='Churn', title_name='是否开通电话服务业务与是否流失的关系')
plot_bar(input_col='MultipleLines', target_col='Churn', title_name='是否开通多线业务与是否流失的关系')
plot_bar(input_col='OnlineSecurity', target_col='Churn', title_name='是否开通网络安全服务与是否流失的关系')
plot_bar(input_col='InternetService', target_col='Churn', title_name='是否开通互联网服务与是否流失的关系')
```

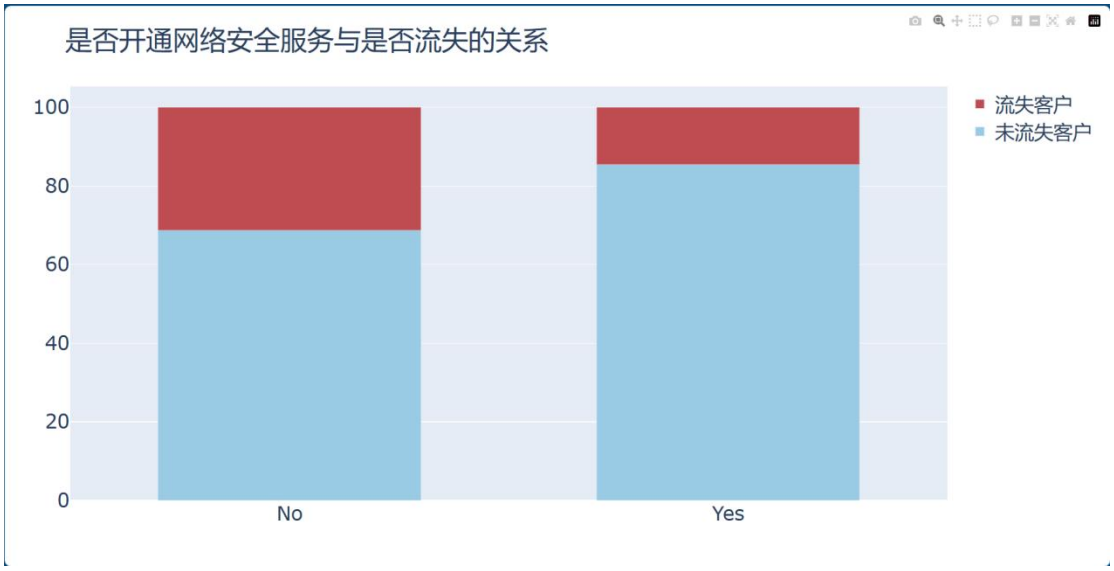
表 5 类别型变量取值与是否流失的关系代码

图 6 是否有配偶与是否流失的关系:



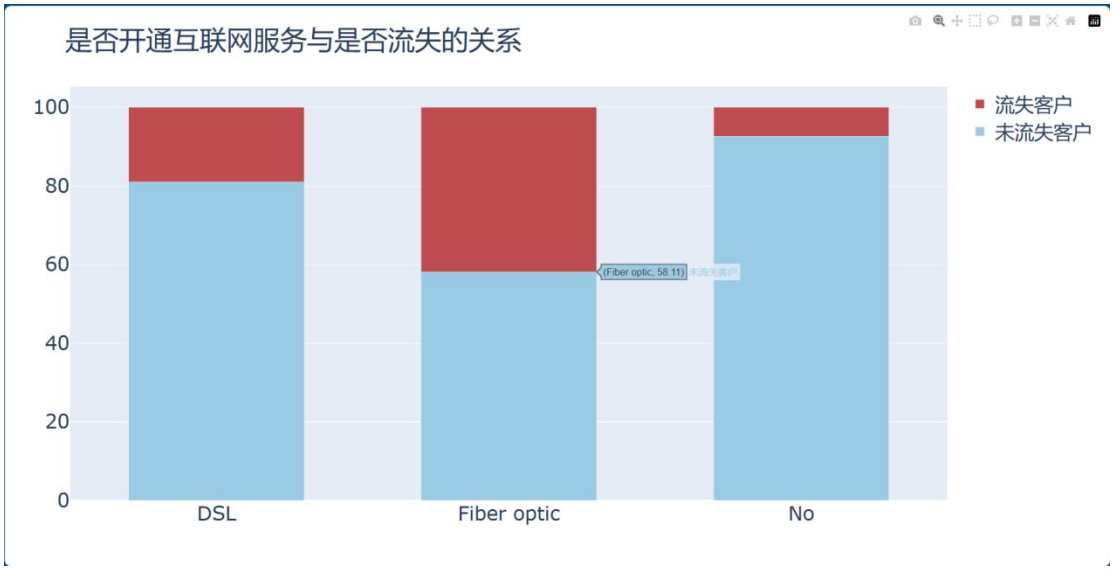
从配偶情况来看，没有配偶的用户流失率 32.98%要远远高于有配偶的用户 19.72%。

图 7 是否开通网络安全服务与是否流失的关系



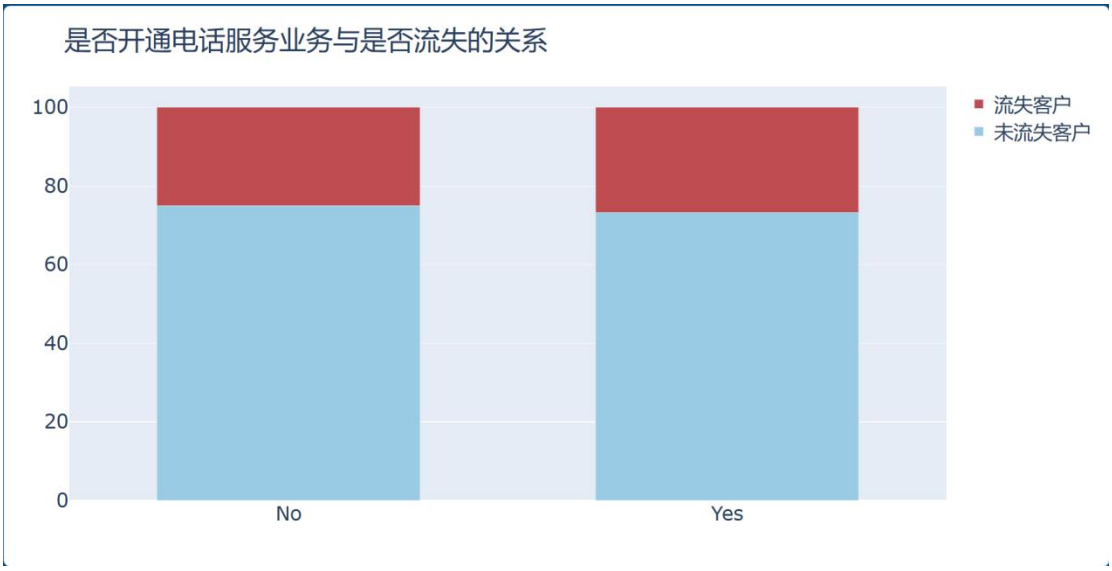
与配偶情况类似，没有开通网络安全服务的流失率 31.37%高于开通网络安全服务的流失率 14.64%.

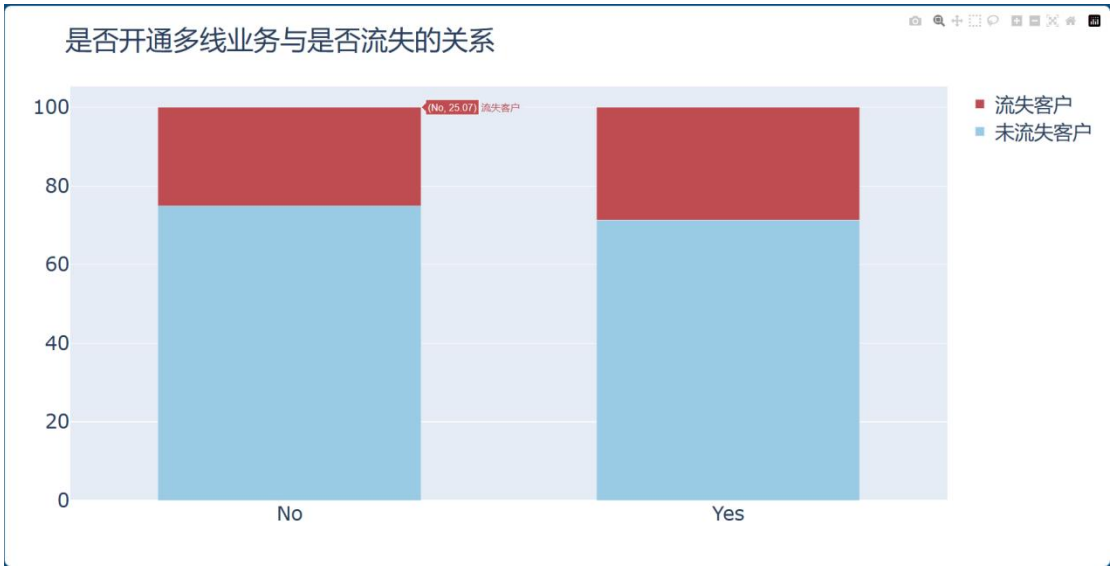
图 8 是否开通互联网服务与是否流失的关系



可以看出，是否开通互联网服务对于客户流失率影响很大，DSL(宽带上网)的流失率为19%,未开通互联网服务的流失率仅为 7.43%,而使用光纤服务的客户流失率达到了 41.89%。可以退出互联网服务对于传统的电信服务带来了巨大冲击。

图 9 是否开通电话服务、多线业务与是否流失的关系





可以看出是否开通多线业务和电话服务业务对于用户流失率影响不大。

c) 查看数值型变量取值与流失的关系。

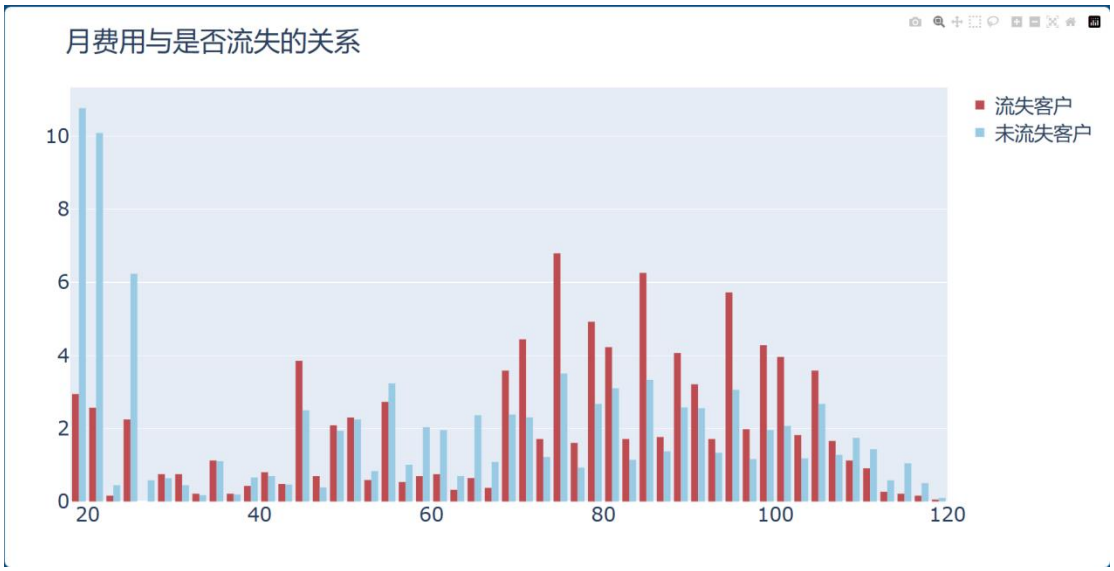
数值型变量与流失关系代码如表 6 所示：

```
# 数值型变量取值与是否流失的关系代码
def plot_histogram(input_col: str, title_name: str):
    churn_num = df[df['Churn'] == 'Yes'][input_col]
    not_churn_num = df[df['Churn'] == 'No'][input_col]

    # 图形轨迹
    trace0 = go.Histogram(x=churn_num,
                          bingroup=25,
                          histnorm='percent',
                          name='流失客户',
                          marker=dict(color='rgb(191,76,81)')
                          )
    trace1 = go.Histogram(x=not_churn_num,
                          bingroup=25,
                          histnorm='percent',
                          name='未流失客户',
                          marker=dict(color='rgb(154,203,228)')
                          )

    data = [trace0, trace1]
    layout = go.Layout(title=title_name, font=dict(size=26))

    fig = go.Figure(data=data, layout=layout)
    py.offline.plot(fig, filename=f'./html/{title_name}.html', auto_open=False)


plot_histogram(input_col='MonthlyCharges', title_name='月费用与是否流失的关系')
```

```
plot_histogram(input_col='TotalCharges', title_name='总费用与是否流失的关系')
```

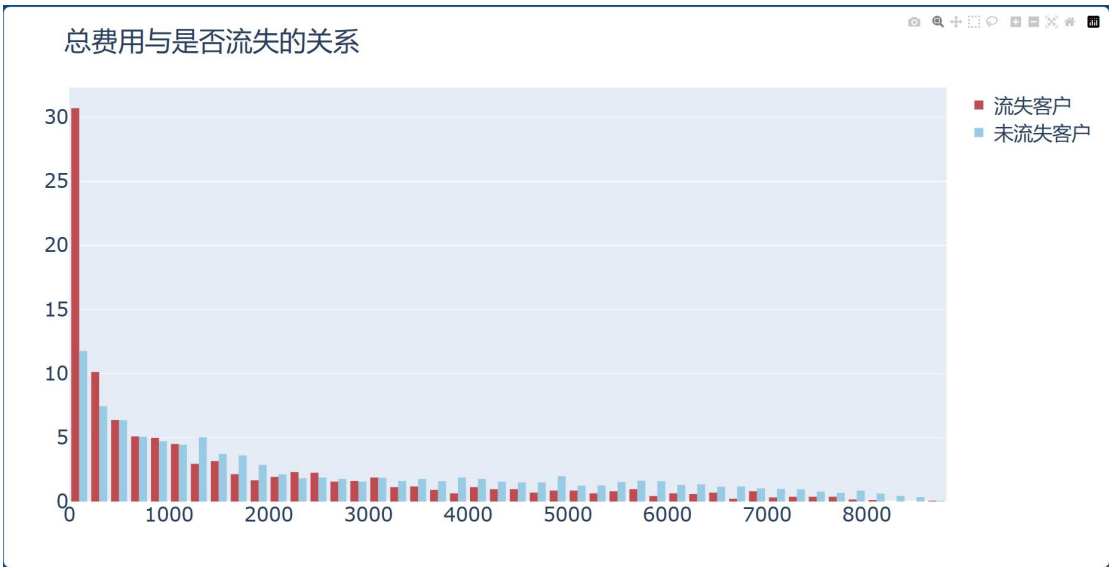
表 6 数值型变量取值与是否流失的关系代码

图 10 月费用与是否流失的关系



整体来看，随着月费用的增加，流失用户的比例呈现高高低低的变化，月消费 80-100 元的用户相对较高。

图 11 总费用与是否流失的关系



总体来看，随着总费用的增加，客户占比越来越少，但是同一数值下流失用户比例也在降低。这说明消费金额在 0-2000 元的客户比较多，而且消费金额越多的客户群体中流失客户占比也较少。

d) 探索数值型数据的相关性

探索数值型数据的相关性代码如表 7 所示：

```
# 探索数值型变量相关性
# 中文显示问题
import matplotlib

matplotlib.rc("font", family='SimHei')
```

```
# 仅选择数值型列计算相关性
num_cols = df.select_dtypes(include=[np.number]).columns
df_num = df[num_cols]

plt.figure(figsize=(8, 6))
corr = df_num.corr()
sns.heatmap(corr, linewidths=0.1, cmap='tab20c_r', annot=True)
plt.title('数值型属性的相关性', fontdict={'fontsize': 'xx-large', 'fontweight': 'heavy'})
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show()
```

表 7 探索数值型数据的相关性代码

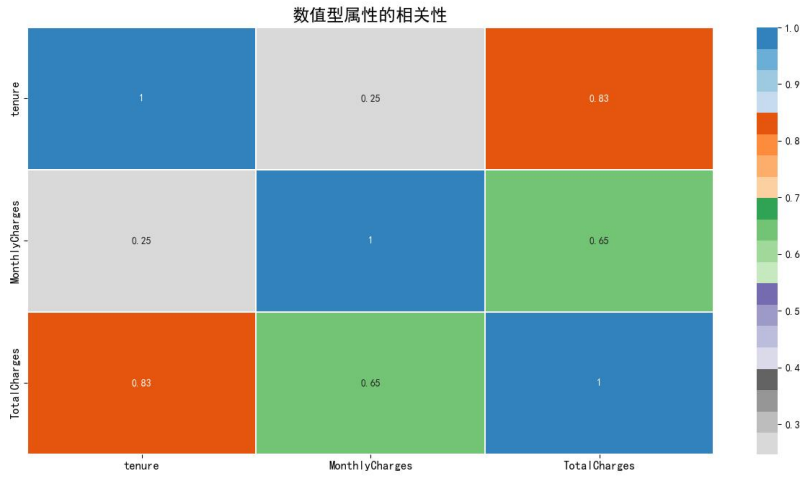


图 12 数值型属性的相关性矩阵图

从相关性矩阵图可以看出，用户的在网时长（tenture）和总费用（TotalCharges）呈现高度相关，往来期间越长，则总费用越高。月消费和总消费呈现显著相关。

（4）建模前处理

满足 python 建模需要，需要对数据做以下处理。

对于分类变量，编码为 0 和 1;

对于多分类变量，进行哑变量转换；

对于数值型变量，部分模型如 KNN、神经网络、Logistic 需要进行标准化处理。

```
# 对于二分类变量，编码为 0 和 1;
# 对于多分类变量，进行 one_hot 编码；
# 对于数值型变量，部分模型如 KNN、神经网络、Logistic 需要进行标准化处理。
# 建模数据
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler

df_model = df
```

```
Id_col = ['customerID']
target_col = ['Churn']
# 分类型
cat_cols = df_model.nunique()[df_model.nunique() < 10].index.tolist()
# 二分类属性
binary_cols = df_model.nunique()[df_model.nunique() == 2].index.tolist()
# 多分类属性
multi_cols = [i for i in cat_cols if i not in binary_cols]
# 数值型
num_cols = [i for i in df_model.columns if i not in cat_cols + Id_col]
# 标准化处理
scaler = StandardScaler()
df_model[num_cols] = scaler.fit_transform(df_model[num_cols])
# 二分类-标签编码
le = LabelEncoder()
for i in binary_cols:
    df_model[i] = le.fit_transform(df_model[i])
# 多分类-哑变量转换
df_model = pd.get_dummies(data=df_model, columns=multi_cols)
df_model.head()
```

表 8 建模前数据处理的代码

（5）特征筛选

基于过滤法选取特征相关性最高的 20 个特征,为后续模型建模做准备。

```
# 使用统计检定方式进行特征筛选。
# from sklearn import feature_selection
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

X = df_model.copy().drop(['customerID', 'Churn'], axis=1)
y = df_model[target_col]
fs = SelectKBest(score_func=f_classif, k=20)
X_train_fs = fs.fit_transform(X, y)
X_train_fs.shape



def SelectName(feature_data, model):
    scores = model.scores_
    indices = np.argsort(scores)[::-1]
    return list(feature_data.columns.values[indices[0:model.k]])



# 输出选择变量名称
```

```
print(SelectName(X, fs))
fea_name = [i for i in X.columns if i in SelectName(X, fs)]
X_train = pd.DataFrame(X_train_fs, columns=fea_name)
X_train.head()
```

表 9 特征筛选的代码

输出结果如下图所示，可以看出，经统计检定方式筛选后的变量共有 20 个。

```
['Contract_Month-to-month', 'tenure', 'tenure_group_Tenure_1', 'InternetService_Fiber optic', 'Contract_Two year', 'PaymentMethod_Electronic check', 'InternetService_No', 'tenure_group_Tenure_over_5', 'TotalCharges', 'MonthlyCharges', 'PaperlessBilling', 'Contract_One year', 'OnlineSecurity', 'TechSupport', 'Dependents', 'SeniorCitizen', 'Partner', 'PaymentMethod_Credit card (automatic)', 'InternetService_DSL', 'PaymentMethod_Bank transfer (automatic)']
```

图 13 筛选后的变量

1.4 模型构建

本案例模型构建工作步骤如下。

（1）建立基准模型

```
# 建模
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import cross_val_score

# 实例模型
logit = LogisticRegression()

knn = KNeighborsClassifier(n_neighbors=5)

svc_lin = SVC(kernel='linear', random_state=0, probability=True)

svc_rbf = SVC(kernel='rbf', random_state=0, probability=True)

mlp_model = MLPClassifier(hidden_layer_sizes=(8,), alpha=0.05, max_iter=50000,
                          activation='logistic', random_state=0)

gnb = GaussianNB()

decision_tree = DecisionTreeClassifier(criterion='entropy', max_depth=5, random_state=0)
```

```
rfc = RandomForestClassifier(n_estimators=100, random_state=0)

lgbm_c = LGBMClassifier(boosting_type='gbdt', n_estimators=100, random_state=0)

xgc = XGBClassifier(n_estimators=100, eta=0.02, max_depth=15, random_state=0, learning_rate=0.001)
```

表 10 建立基准模型的代码

导入基准模型并定义、设置好参数，为后面的十折交叉验证做准备。

（2）十折交叉验证并比较效果

```
# 模型列表
models = [logit, knn, svc_lin, svc_rbf, mlp_model, gnb,
          decision_tree, rfc, lgbm_c, xgc]

model_names = ["Logistic Regression", "KNN Classifier", "SVM Classifier Linear",
               "SVM Classifier RBF", "MLP Classifier", "Naive Bayes",
               "Decision Tree", "Random Forest Classifier", "LGBM Classifier",
               "XGBoost Classifier"]

scoring = ['accuracy', 'recall', 'precision', 'f1', 'roc_auc']
# 多折交叉验证
results = pd.DataFrame(columns=['Model'] + list(scoring))

for i, model in enumerate(models):

    name = model_names[i]

    cv_results = cross_validate(model, X, y, cv=10,
                                scoring=scoring, return_train_score=False)
    mean_scores = np.zeros(len(scoring))
    for i, score in enumerate(scoring):
        mean_scores[i] = np.mean(cv_results[f'test_{score}'])

    temp_df = pd.DataFrame([{
        'Model': name,
        **dict(zip(scoring, mean_scores))
    }], columns=results.columns)

    results = pd.concat([results, temp_df], ignore_index=True)

table = ff.create_table(np.round(results, 4))
py.offline.iplot(table)
```

表 11 十折交叉验证的代码

结果如图 14 所示

Model	accuracy	recall	precision	f1	roc_auc
Logistic Regression	0.8065	0.535	0.6706	0.5948	0.8477
KNN Classifier	0.7719	0.5249	0.5789	0.5502	0.7874
SVM Classifier Linear	0.7966	0.4644	0.6691	0.5479	0.8307
SVM Classifier RBF	0.8013	0.4847	0.6757	0.5637	0.8001
MLP Classifier	0.8032	0.5388	0.6589	0.5927	0.8455
Naive Bayes	0.7466	0.786	0.5157	0.6226	0.8318
Decision Tree	0.7907	0.5838	0.6125	0.5963	0.8326
Random Forest Classifier	0.7976	0.5008	0.659	0.5687	0.8271
LGBM Classifier	0.7972	0.519	0.6501	0.5765	0.8371
XGBoost Classifier	0.7342	0.0	0.0	0.0	0.8165

图 14 模型处理结果表

结果如图 14 所示，假设我们关注 roc 指标，从模型表现效果来看，Naive Bayes 效果最好。

（3）针对不平衡数据集采用 SMOTE 算法优化

```
from imblearn.over_sampling import SMOTE

smo = SMOTE(random_state=42)
X_smo, y_smo = smo.fit_resample(X, y)
```

表 12 SMOTE 算法优化的代码

在数据处理中我们发现，数经过初步清洗之后的数据集大小为 7032 条记录，其中流失客户为 1869 条，占比 26.6%，未流失客户占比 73.4%。(略微不平衡)

不平衡分类的一个问题是少数类的样本太少，模型无法有效地学习决策边界。我采用 SMOTE 模型对少数类中的样本进行过采样来优化处理。

结果如图 15 所示，经过 SMOTE 算法处理过的模型各项指标明显上升。

Model	accuracy	recall	precision	f1	roc_auc
Logistic Regression	0.8206	0.8404	0.8074	0.8209	0.9076
KNN Classifier	0.8226	0.9171	0.7714	0.8376	0.8924
SVM Classifier Linear	0.8186	0.8355	0.8068	0.8182	0.9074
SVM Classifier RBF	0.821	0.8443	0.8062	0.8227	0.9019
MLP Classifier	0.8194	0.8301	0.8117	0.8172	0.9072
Naive Bayes	0.7708	0.8195	0.7469	0.7813	0.8448
Decision Tree	0.7737	0.8114	0.7547	0.7815	0.8543
Random Forest Classifier	0.8555	0.8838	0.8361	0.8572	0.9302
LGBM Classifier	0.8338	0.8584	0.8175	0.8346	0.9182
XGBoost Classifier	0.803	0.8295	0.7871	0.8064	0.8787

图 15 SMOTE 优化后的模型处理结果表

1.5 模型评估

（1）使用 LightGBM 实现内置集成

```
# 导入评估指标计算函数
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score, roc_auc_score
from lightgbm import LGBMClassifier, log_evaluation, early_stopping

# 评估指标列表
scoring = ['accuracy', 'recall', 'precision', 'f1', 'roc_auc']

# 数据处理
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

```
# 模型训练
callbacks = [log_evaluation(period=100), early_stopping(stopping_rounds=30)]
gbm = LGBMClassifier(num_leaves=31, learning_rate=0.05, n_estimators=100)
gbm.fit(X_train, y_train, eval_set=[(X_test, y_test)], callbacks=callbacks)

# 模型保存
joblib.dump(gbm, 'model.pkl')

# 预测和评估
y_pred = gbm.predict(X_test, num_iteration=gbm.best_iteration_)

result = {}
for metric in scoring:
    if metric == 'accuracy':
        result[metric] = accuracy_score(y_test, y_pred)
    elif metric == 'recall':
        result[metric] = recall_score(y_test, y_pred)
    elif metric == 'precision':
        result[metric] = precision_score(y_test, y_pred)
    elif metric == 'f1':
        result[metric] = f1_score(y_test, y_pred)
    elif metric == 'roc_auc':
        result[metric] = roc_auc_score(y_test, y_pred)
# 生成结果表
results = pd.DataFrame([result], columns=scoring)
print(results)
```

表 13 使用 LightGBM 实现内置集成的代码

通过引入 LightGBM 的 early stopping 回调以及 log evaluation 功能,监控模型在训练集和验证集上的表现,避免过拟合。利用 sklearn 接口训练模型,计算多指标如 Accuracy、Recall、Precision 等指标进行评估,通过 panda 表格输出评价结果以一目了然了解模型效果。最后记录特征重要性,为后续参数优化提供依据。

结果：

```
[59] valid_0's binary_logloss: 0.40238
    accuracy     recall  precision        f1    roc_auc
0   0.821957   0.556818   0.674931   0.610212   0.733644
```

表 14 使用 LightGBM 实现内置集成的输出结果

可见，集成学习结果和未使用 SMOTE 处理前的 LGBM classifier 模型结果相比有较大提升。

（2）使用 Bagging 算法实现集成

```
# 导入评估指标计算函数
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score, roc_auc_score
```

```
from sklearn.ensemble import BaggingClassifier

# 评估指标列表
scoring = ['accuracy', 'recall', 'precision', 'f1', 'roc_auc']

# 数据处理
X_train, X_test, y_train, y_test = train_test_split(X, y)

# 模型训练
bagging_clf = BaggingClassifier(base_estimator=DecisionTreeClassifier(),
                                n_estimators=100,
                                max_samples=0.7,
                                bootstrap=True)

# 预测和评估
bagging_clf.fit(X_train, y_train)

y_pred = bagging_clf.predict(X_test)

result = {}
for metric in scoring:
    if metric == 'accuracy':
        result[metric] = accuracy_score(y_test, y_pred)
    elif metric == 'recall':
        result[metric] = recall_score(y_test, y_pred)
    elif metric == 'precision':
        result[metric] = precision_score(y_test, y_pred)
    elif metric == 'f1':
        result[metric] = f1_score(y_test, y_pred)
    elif metric == 'roc_auc':
        result[metric] = roc_auc_score(y_test, y_pred)
# 生成结果表
results = pd.DataFrame([result], columns=scoring)
print(results)
```

表 15 使用 Bagging 实现内置集成的输出结果

同理，通过 Baggging 算法进行集成学习,计算多指标如 Accuracy、Recall、Precision 等指标进行评估,通过 panda 表格输出评价结果

结果：

	accuracy	recall	precision	f1	roc_auc
0	0.786121	0.519751	0.632911	0.570776	0.703102

表 16 使用 Bagging 实现内置集成的输出结果

由结果可见，Bagging 算法实现的集成学习的各项指标明显低于 LGBM classifier 模型实现的集成学习,LightGBM 实现内置集成在分类效果上更好。

（3）绘制 ROC 曲线

```
# 绘制 ROC 曲线
from matplotlib import pyplot as plt
from sklearn.metrics import roc_curve, auc

fprs = []
tprs = []
aucs = []

# 数据处理
X_train, X_test, y_train, y_test = train_test_split(X, y)



def roc_img(model, X_train, X_test, y_train, y_test, name):
    model.fit(X_train, y_train)
    y_score = model.predict_proba(X_test)[:, 1]

    fpr, tpr, thresholds = roc_curve(y_test, y_score)
    roc_auc = auc(fpr, tpr)

    print(roc_auc)

    fprs.append(fpr)
    tprs.append(tpr)
    aucs.append(roc_auc)



models = [logit, knn, svc_lin, svc_rbf, mlp_model, gnb, decision_tree, rfc, lgbm_c, xgc]
names = ["Logistic Regression", "KNN Classifier", "SVM Clissifier Linear",
         "SVM Classifier RBF", "MLP Classifier", "Naive Bayes", "Decision Tree",
         "Random Forest Classifier", "LGBM Classifier", "XGBoost Classifier"]

for i in range(10):
    roc_img(models[i], X_train, X_test, y_train, y_test, names[i])
    plt.plot(fprs[i], tprs[i], lw=1.5, label="%s, AUC=%.3f" % (names[i], aucs[i]))

plt.xlabel("FPR", fontsize=15)
plt.ylabel("TPR", fontsize=15)

plt.title("ROC")
plt.legend(loc="lower right")
plt.show()
```

表 17 绘制 ROC 曲线的代码

结果：

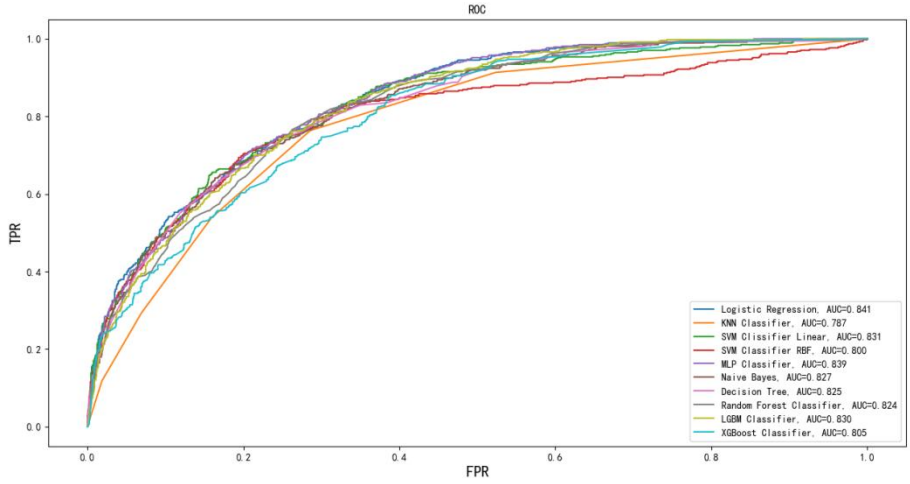


图 16 ROC 曲线图

同样，如果对数据集进行 SMOTE 处理，各模型的 AUC 值也会显著增大。

```
# 数据处理
smo = SMOTE(random_state=42)
X_smo, y_smo = smo.fit_resample(X, y)
X_train, X_test, y_train, y_test = train_test_split(X_smo, y_smo)
```

表 18 绘制 ROC 曲线过程中不平衡数据处理

结果：

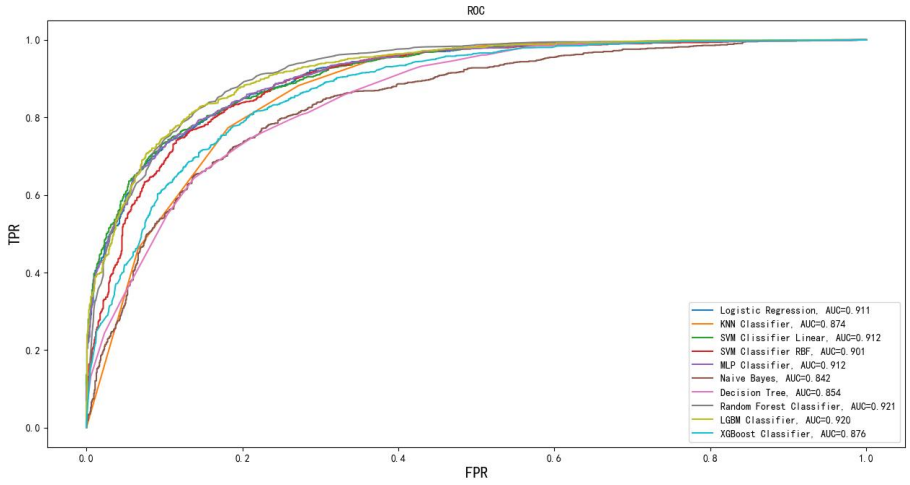


图 17 SMOTE 优化后的 ROC 曲线图

总结一下,在通信客户流失预警模型实验中绘制 ROC 曲线的主要意义有:

1. 可以更直观地展示模型在不同阈值下的识别能力,帮助选择最优阈值。
2. AUC 可以量化不同模型的分类能力,选择效果最佳模型。
3. 可视化不同算法的优劣,评价模型效果。
4. 为后期上线提供选择最佳阈值的依据,平衡识别率和覆盖率。
5. 提供预测结果与流量管理和下游业务调度的重要参考。

总体来说,ROC 曲线可以从多角度进行模型评价,为流失预警任务提供重要的决策支持。

（4）对决策树参数调优

```
# 我们也可以对模型进行进一步优化，比如对决策树参数进行调优。
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

parameters = {'splitter': ('best', 'random'),
                    'criterion': ("gini", "entropy"),
                    "max_depth": [*range(3, 20)],
                    }

clf = DecisionTreeClassifier(random_state=25)

GS = GridSearchCV(clf, parameters, scoring='f1', cv=10)
GS.fit(X_train, y_train)
print(GS.best_params_)
print(GS.best_score_)
clf = GS.best_estimator_
test_pred = clf.predict(X_test)
print('测试集：n', classification_report(y_test, test_pred))

# In[12]:


# 将这棵树画出来
import graphviz

from pydotplus.graphviz import graph_from_dot_data
from sklearn.tree import export_graphviz

part_DT = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)
part_DT.fit(X_train, y_train)

dot_data = tree.export_graphviz(decision_tree=part_DT, max_depth=3,
                                        out_file=None,
                                        #
feature_names=X_train.columns,
                                        feature_names=X_train.columns,
                                        class_names=['not_churn', 'churn'],
                                        filled=True,
                                        rounded=True
                                        )
graph = graphviz.Source(dot_data)
```

```
graph = graph_from_dot_data(dot_data)    # Create graph from dot data

graph.write_png('./决策树.png')    # Write graph to PNG image

# In[13]:


# 输出决策树属性重要性排序
import plotly.figure_factory as ff

import plotly as py

imp = pd.DataFrame(zip(X_train.columns, clf.feature_importances_))

imp.columns = ['feature', 'importances']

imp = imp.sort_values('importances', ascending=False)

imp = imp[imp['importances'] != 0]

table = ff.create_table(np.round(imp, 4))

py.offline.iplot(table)
```

表 19 优化决策树并输出属性重要性排序代码

结果 1：

```
{'criterion': 'entropy', 'max_depth': 5, 'splitter': 'best'}
0.6014522720158608
测试集：n                    precision    recall  f1-score   support

           0       0.84      0.88      0.86      1033
           1       0.62      0.53      0.57       374

    accuracy                           0.79      1407
   macro avg       0.73      0.71      0.71      1407
weighted avg       0.78      0.79      0.78      1407
```

表 20 优化决策树并输出属性重要性排序代码

此代码中的 GridSearchCV 用于对决策树算法进行参数优化:

1. 定义需要优化的参数集合,包括决策树的划分标准、最大深度和节点选择策略等参数。
2. 使用 GridSearchCV 进行参数竞赛,遍历所有参数组合,选择 F1 得分最高的最佳参数组合。
3. 记录最优参数和 F1 得分。
4. 使用最优参数重新训练决策树模型,在测试集上进行预测。

5. 计算混淆矩阵和分类报告,评估使用优化参数后的决策树模型效果。

输出结果显示:

- 最优参数为 criteria=entropy,最大深度 5,节点选择 best
- F1 得分 0.6015,说明优化后模型效果有所提升
- 测试集分类报告 auc、recall、precision 指标都有一定提升

结果 2：

feature	importances
Contract_Month-to-month	0.5369
tenure	0.1441
InternetService_Fiber optic	0.1075
TotalCharges	0.0726
MonthlyCharges	0.0618
PaymentMethod_Electronic check	0.0174
Contract_Two year	0.0166
InternetService_DSL	0.0165
InternetService_No	0.0111
TechSupport	0.007
Contract_One year	0.0037
PaymentMethod_Credit card (automatic)	0.0034
PaymentMethod_Bank transfer (automatic)	0.0013

图 18 决策树属性重要性排序

从图中可以得出对模型训练影响的权重最大的几个属性，比如 Contract_Month-to-Month,tenure.从中可以进一步分析影响客户流失的关键因素。

结果 3：

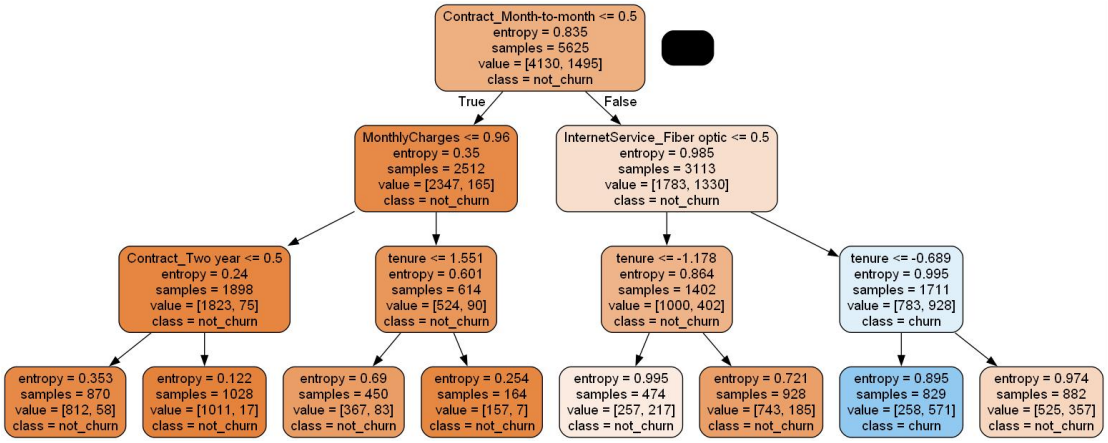


图 19 决策树

（5）后续优化方向

数据方面:

分析客户流失率分布,采用过采样或者欠采样来处理流失与保留样本不平衡问题

增加其他维度数据集如客户属性、交互行为等深入挖掘客户特征

特征方面:

对原特征进一步相关性分析和筛选,去除多重共线特征

尝试基于原属性构建新特征,如客户活跃程度

尝试将分类和回归特性整合为新的特征

算法方面:
选择 LightGBM 和 XGBoost 等更优算法进行参数优化,比如学习率、树叶节点等
尝试集成学习方法,整合多个基学习器提升预测性能
调整阈值寻找平衡识别率和覆盖率的最优点

模型评估方面:
增加交叉验证操作避免过拟合
多指标评价并绘制学习曲线观察泛化能力
对比 AUC 等指标选择最优模型上线

1.6 模型部署

本实验通信客户流失预警模型的部署工作如下:
1. 根据流失预测任务的需求,将训练好的模型参数和结构保存到产品服务端,使用预训练模型直接进行预测服务。
2. 考虑模型预测性能和服务压力,采用更高效的模型如 LightGBM 或 XGBoost 部署,降低响应时间。
3. 为了实时性,需要周期性使用新收集的客户数据重新训练模型,保证预测效果随时保持实时最优。
4. 部署在线预测服务接口,业务系统可以通过接口调用获取每个客户的流失风险置信度,并提供给下游系统使用。
5. 监控部署模型在线服务的性能和健壮性,及时排查和解决应用故障。
6. 定期评估模型效果,根据业务需求及数据变化重新优化模型 installed 应用。
7. 提供模型解释功能,帮助业务人员理解预测结果背后的决策依据。
8. 保护客户隐私,提供安全可靠的服务系统。
将测试结果上线服务,实现成果产品化应用,是实验工作的重要起点。

2、实验小结

主要工作小结
通过案例数据挖掘得到的成果、对客户保留计划的建议
本实验过程中的创新点

2.1 主要工作小结

本实验完成了通信客户流失预警模型从数据获取、数据处理、建模优化、参数调优到模型评估和部署的全流程。

首先,对原始数据集进行数据清洗,处理缺失值和不具备表达能力的特征。然后对变量类型进行统计,提取数值型和类别型变量。

其次,通过关系可视化分析变量相关性,并对目标变量和特征变量之间的关系进行探索。然后使用统计检验方法进行特征选择,选择影响较大的特征。

接着,对数据进行建模处理,对类别型变量进行编码,对数值型变量进行标准化。然后利用十折交叉验证评估多种算法的效果,选择 lgbm 模型进行训练。

然后,对决策树模型进行参数优化,找到最佳参数对模型效果的提升。最后对模型结果进行可视化,包括决策树结构和属性重要性。

最后,存档最优模型,并绘制 ROC 曲线对比不同模型的性能。本实验全面而系统地解决了客户流失预测问题,为后续实际应用提供了可靠的技术支撑。

通过学习已有客户的数据，掌握了对用户流失有关键影响的特征。预测分析使用客户流失预测模型，通过评估客户流失的风险倾向来预测客户流失。由于这些模型生成了一个流失概率排序名单，对于潜在的高概率流失客户，可以有效地实施客户保留营销计划。

2.2 数据挖掘成果与建议

2.2.1 挖掘成果

本实验分析了数据集中的 7043 笔客户资料，每笔客户资料包含 21 个字段，其中 1 个客户 ID 字段，19 个输入字段及 1 个目标字段-Churn（Yes 代表流失，No 代表未流失），输入字段主要包含以下三个维度指标：用户画像指标、消费产品指标、消费信息指标。

首先实验研究了整体流失情况，得知了整体用户中流失用户占比较多，73.4%.

然后对 19 个指标进行分析，研究它们对用户流失的影响，比如：

1.从配偶情况来看，没有配偶的用户流失率 32.98%要远远高于有配偶的用户 19.72%。

2.整体来看，随着月费用的增加，流失用户的比例呈现高高低低的变化，月消费 80-100 元的用户相对较高。

3.总体来看，随着总费用的增加，客户占比越来越少，但是同一数值下流失用户比例也在降低。这说明消费金额在 0-2000 元的客户比较多，而且消费金额越多的客户群体中流失客户占比也较少。

实验还研究得出了数值型指标之间的关系：从相关性矩阵图可以看出，用户的在网时长（tenture）和总费用（TotalCharges）呈现高度相关，往来期间越长，则总费用越高。月消费和总消费呈现显著相关。

然后我们构建了多种模型，又用了多种指标去评估模型，优化模型。最后得出随机森林模型的效果最好的结论，同时我们研究了对用户流失影响占比权重最大的几个特征：Contract_Month-to-month(签订合同方式),tenure(在网时长),Internet Service_Fiber optic(是否开通互联网服务).

2.2.1 建议

综上所述，我建议客户保留计划使用随机森林模型，通过评估客户流失的风险倾向来预测客户流失。

对于没有配偶的用户群体,应加强留意和服务。可以定期采取问候邮件或短信,了解客户需求,增强粘性。必要时提供一定单身用户专享优惠。

对于月消费在 80-100 元的用户群体,应重点开展留存工作。可通过优惠活动降低此消费区间用户的实际费用,增加他们的消费体验。

对于总费用在 0-2000 元的大量客户群体,采取分层留存策略。费用较低用户给予免费套餐等好处,费用中高用户提供更多增值服务。

强化月到月合同用户的绑定力。推出长期套餐让用户受益更多。同时加强这部分客户的沟通与顾问服务。

鼓励用户增开各类增值业务。Internet Service_Fiber optic 用户流失率较低,说明增开业务有利于留存。

根据用户在网时长给予等级达成奖励,激发用户活跃度。时长越长,优惠越高。

分析影响力最大特征下的细分客户群,进行定制化留存。

2.3 实验过程中的创新点

关于本次实验的创新点,我进行了以下几点改进:

1. 数据清洗方面,我额外检查了**特征缺失值情况**和**数值型特征中的异常点**,绘制了**箱线图**，以进一步清洗数据质量。
2. 探索性分析方面,我选择了与原实验中**不同的 5 个类别型特征**和**年费用**这一数值型特征,分析其与用户流失的关系,使得结果更全面系统。
3. 模型构建阶段,我使用**十折交叉验证**替代了原先的训练集测试集划分方法,利用全部数据信息,有效提升模型评估的稳定性与可靠性。
4. 我还采用 **SMOTE** 进行**样本平衡处理**,有效解决了因为目标类别不平衡给模型带来的影响,进一步优化了模型性能。
5. 在模型评估阶段,我选择使用**集成学习**性能极佳的 **LightGBM 算法和传统的 Bagging 算法**,，并比较它们的分类效果，实现内置的变量重要性计算等功能,使分析过程更科学系统化。
6.我在绘制 **ROC 图**时对输入数据同样进行了 SMOTE 算法的处理，得到了 **AUC 值**更高的结果.

总体来说,我在原有代码基础上进行了一些数据处理、算法选择与模型构建等方面的改进,发挥了创新精神,最终实现了更优的客户流失预测效果。