通信客户流失预警模型案例分析

1. 环境介绍

windows 10

conda 4.7.12（Anaconda3-2019.10-Windows-x86\_64）

python 3.7.4

PyCharm 2022.3.3

jupyter notebook

1. 业务理解

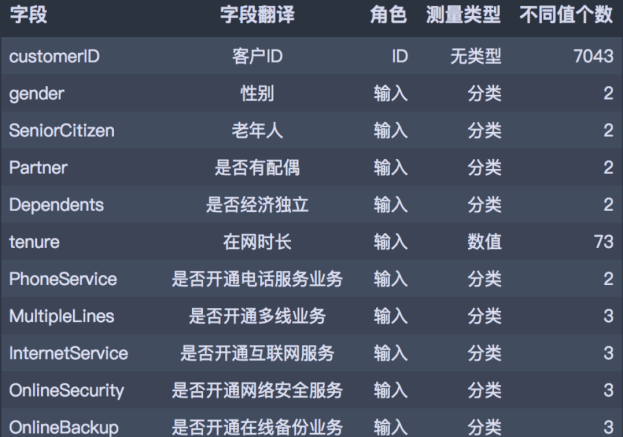
流失客户是指那些曾经使用过产品或服务，由于对产品失去兴趣等种种原因，不再使用产品或服务的顾客。

电信服务公司、互联网服务提供商、保险公司等经常使用客户流失分析和客户流失率作为他们的关键业务指标之一，因为留住一个老客户的成本远远低于获得一个新客户。

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，他们可以有效地实施客户保留营销计划。

1. 数据理解

此次分析数据来自于IBM Sample Data Sets，统计自某电信公司一段时间内的消费数据。共有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。字段的具体说明如下：

1. 数据准备

# 数据处理

import pandas as pd

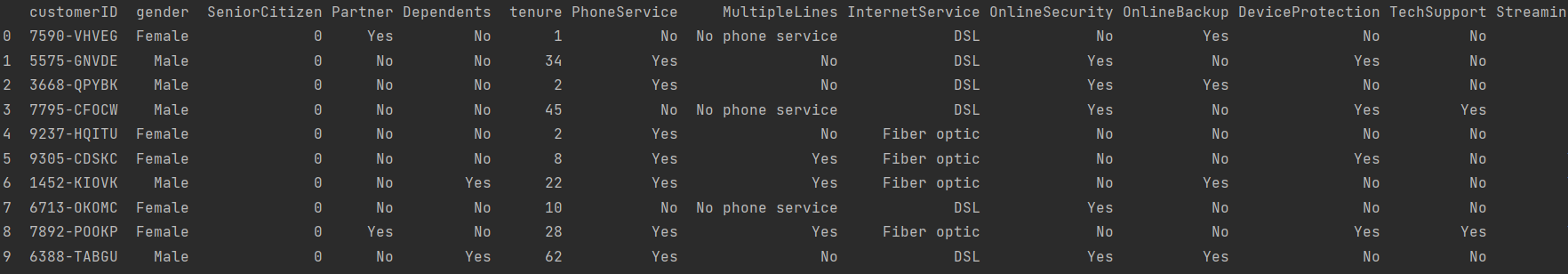
import numpy as np

#读入数据集

df=pd.read\_csv("Telco-Customer-Churn.csv")

print(df.head(10).to\_string())

代码运行如下：



# 数据初步清洗

# 首先进行初步的数据清洗工作，包含错误值和异常值处理，并划分类别型和数值型字段类型，

# 其中清洗部分包含：OnlineSecurity、OnlineBackup、DeviceProtection、TechSupport、StreamingTV、StreamingMovies：

# 错误值处理 TotalCharges：异常值处理 tenure：自定义分箱

# 错误值处理

C\_olumns=['OnlineSecurity','OnlineBackup','DeviceProtection','TechSupport','StreamingTV','StreamingMovies']

for i in C\_olumns:

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

# 替换SeniorCitizen,Yes:1,No:0

df['SeniorCitizen']=df['SeniorCitizen'].replace({1:'Yes',0:'No'})

# 替换TotalCharges进而对空值进行删除

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

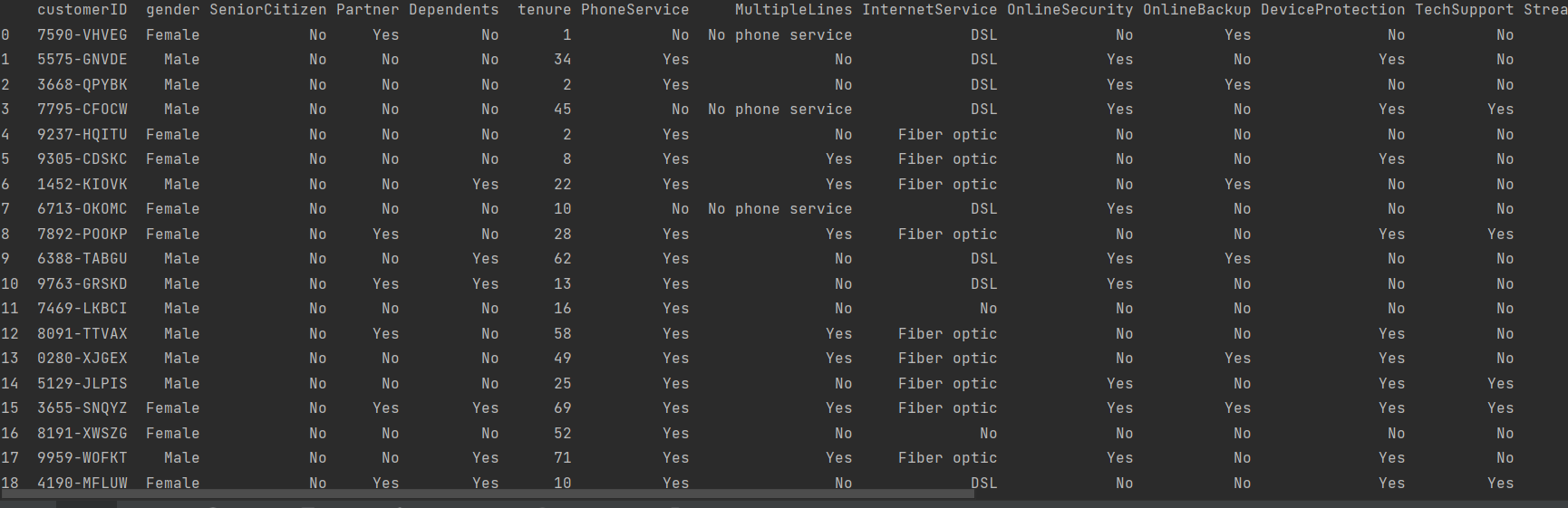
df=df.dropna(subset=['TotalCharges'])

# 重置索引

df.reset\_index(drop=True,inplace=True)

# print(df.head(100).to\_string())

代码运行如下：



# 将TotalCharges列中的字符串转换为浮点数

df['TotalCharges'] = pd.to\_numeric(df['TotalCharges'], errors='coerce')

# 转换tenure,编写函数

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')

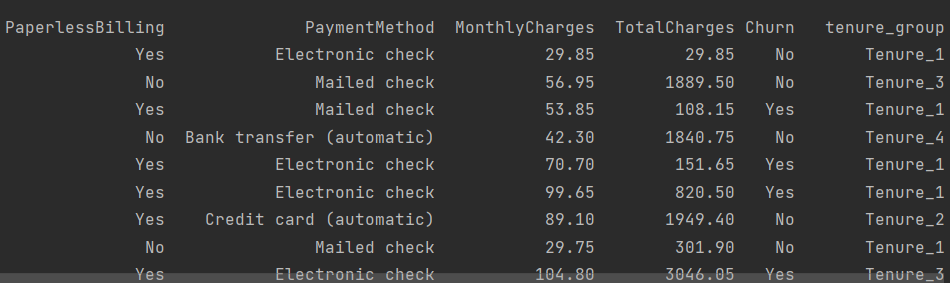
else:

return('Tenure\_over\_5')

df['tenure\_group']=df.tenure.apply(transform\_tenure)

# print(df.head(100).to\_string())

代码运行如下：



# 探索性分析

# 目标变量Churn分布

# 可视化

df['Churn'].value\_counts

trace0 = go.Pie(labels=df[ 'Churn'].value\_counts().index,

values=df[ 'Churn'].value\_counts().values,

hole= 0.5,

rotation= 90,

marker=dict(colors=[ 'rgb(154,203,228)', 'rgb(191,76,81)'],

line=dict(color= 'white', width= 1.3))

)

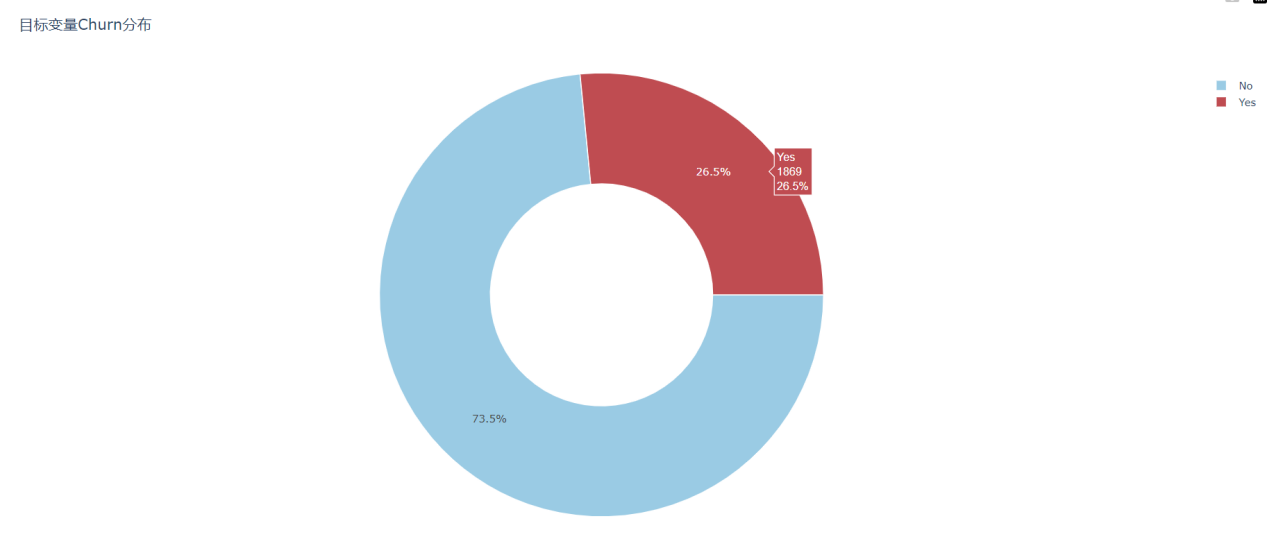
data = [trace0]

layout = go.Layout(title= '目标变量Churn分布')

fig = go.Figure(data=data, layout=layout)

py.offline.plot(fig, filename= '整体流失情况分布.html',auto\_open=False)

效果图如下：



# 性别与是否流失的关系

# 男性和女性在客户流失比例上没有显著差异

a1 = df[(df['Churn']=='Yes')&(df['gender']=='Female')]['Churn'].count()

a\_1 = df[(df['Churn']=='Yes')&(df['gender']=='Male')]['Churn'].count()

a2 = df[(df['Churn']=='No')&(df['gender']=='Female')]['Churn'].count()

a\_2 = df[(df['Churn']=='No')&(df['gender']=='Male')]['Churn'].count()

a1\_p = a1/(a\_1+a1)

a\_1\_p = a\_1/(a\_1+a1)

a2\_p = a2/(a2+a\_2)

a\_2\_p = a\_2/(a2+a\_2)

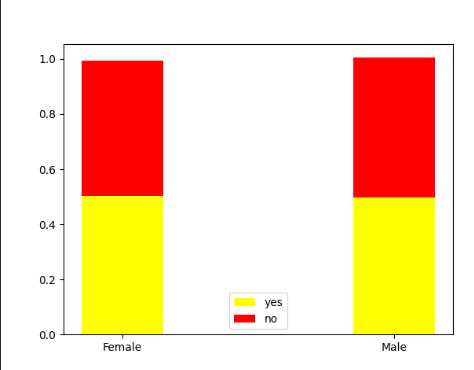
plt.bar(['Female','Male'],height=[a1\_p,a\_1\_p],width=0.3,color='yellow', data=None,label=u'yes')

plt.bar(['Female','Male'],height=[a2\_p,a\_2\_p], width=0.3, bottom=[a1\_p,a\_1\_p],color='red', data=None,label=u'no')

plt.legend(loc='best')

plt.show()

效果图如下：



# 在网时长与是否流失的关系

# 用户的在网时长越长，表示用户的忠诚度越高，其流失的概率越低

a1 = df[(df['Churn']=='Yes')&(df['tenure\_group']=='Tenure\_1')]['Churn'].count()

a\_1 = df[(df['Churn']=='No')&(df['tenure\_group']=='Tenure\_1')]['Churn'].count()

a2 = df[(df['Churn']=='Yes')&(df['tenure\_group']=='Tenure\_2')]['Churn'].count()

a\_2 = df[(df['Churn']=='No')&(df['tenure\_group']=='Tenure\_2')]['Churn'].count()

a3 = df[(df['Churn']=='Yes')&(df['tenure\_group']=='Tenure\_3')]['Churn'].count()

a\_3 = df[(df['Churn']=='No')&(df['tenure\_group']=='Tenure\_3')]['Churn'].count()

a4 = df[(df['Churn']=='Yes')&(df['tenure\_group']=='Tenure\_4')]['Churn'].count()

a\_4 = df[(df['Churn']=='No')&(df['tenure\_group']=='Tenure\_4')]['Churn'].count()

a5 = df[(df['Churn']=='Yes')&(df['tenure\_group']=='Tenure\_5')]['Churn'].count()

a\_5 = df[(df['Churn']=='No')&(df['tenure\_group']=='Tenure\_5')]['Churn'].count()

a6 = df[(df['Churn']=='Yes')&(df['tenure\_group']=='Tenure\_over\_5')]['Churn'].count()

a\_6 = df[(df['Churn']=='No')&(df['tenure\_group']=='Tenure\_over\_5')]['Churn'].count()

a1\_p = a1/(a1+a\_1)

a\_1\_p = a\_1/(a1+a\_1)

a2\_p = a2/(a2+a\_2)

a\_2\_p = a\_2/(a2+a\_2)

a3\_p = a3/(a3+a\_3)

a\_3\_p = a\_3/(a3+a\_3)

a4\_p = a4/(a4+a\_4)

a\_4\_p = a\_4/(a4+a\_4)

a5\_p = a5/(a5+a\_5)

a\_5\_p = a\_5/(a5+a\_5)

a6\_p = a6/(a6+a\_6)

a\_6\_p = a\_6/(a6+a\_6)

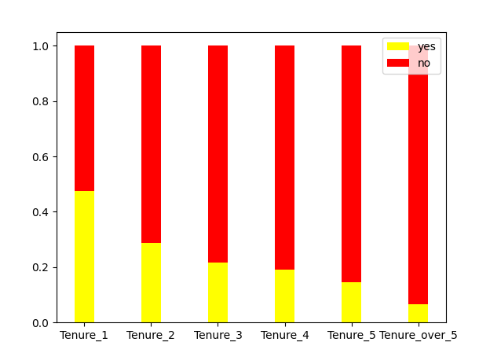
plt.bar(['Tenure\_1','Tenure\_2','Tenure\_3','Tenure\_4','Tenure\_5','Tenure\_over\_5'],height=[a1\_p,a2\_p,a3\_p,a4\_p,a5\_p,a6\_p], width=0.3,color='yellow', data=None,label=u'yes')

plt.bar(['Tenure\_1','Tenure\_2','Tenure\_3','Tenure\_4','Tenure\_5','Tenure\_over\_5'],height=[a\_1\_p,a\_2\_p,a\_3\_p,a\_4\_p,a\_5\_p,a\_6\_p], width=0.3, bottom=[a1\_p,a2\_p,a3\_p,a4\_p,a5\_p,a6\_p],color='red', data=None,label=u'no')

plt.legend(loc='best')

plt.show()

效果图如下：



# 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

df\_model=df

Id\_col=['customerID']

Target\_cil=['Churn']

# 分类型

Category\_cols=df.nunique()[df.nunique()<10].index.tolist()

# 数值型

num\_cols=[i for i in df.columns if i not in Category\_cols +Id\_col]

# 二分类型

binary\_cols=df\_model.nunique()[df\_model.nunique()==2].index.tolist()

# 多分类型

multi\_cols=[i for i in Category\_cols if i not in binary\_cols]

# 二分类标签编码

le=LabelEncoder()

for i in binary\_cols:

df\_model[i]=le.fit\_transform(df\_model[i])

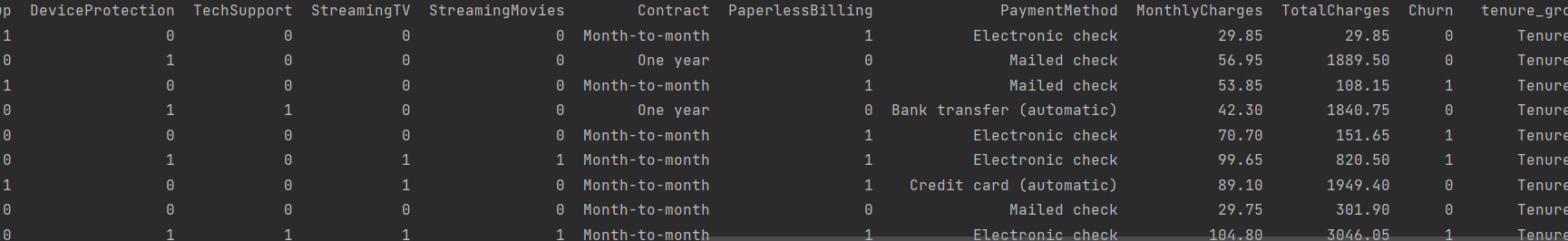
#多分类哑变量变换

df\_model=df\_model.dropna()

df\_model=pd.get\_dummies(data=df\_model,columns=multi\_cols)

print(df.head(100).to\_string())

代码运行如下：



# 使用统计检定方式进行特征筛选。

X = df\_model.copy().drop(['customerID','Churn'], axis=1)

y = df\_model[Target\_cil]

fs = SelectKBest(score\_func=f\_classif, k=20)

y = y.values.ravel()

X\_train\_fs = fs.fit\_transform(X,y)

def SelectName(feature\_data, model):

scores = model.scores\_

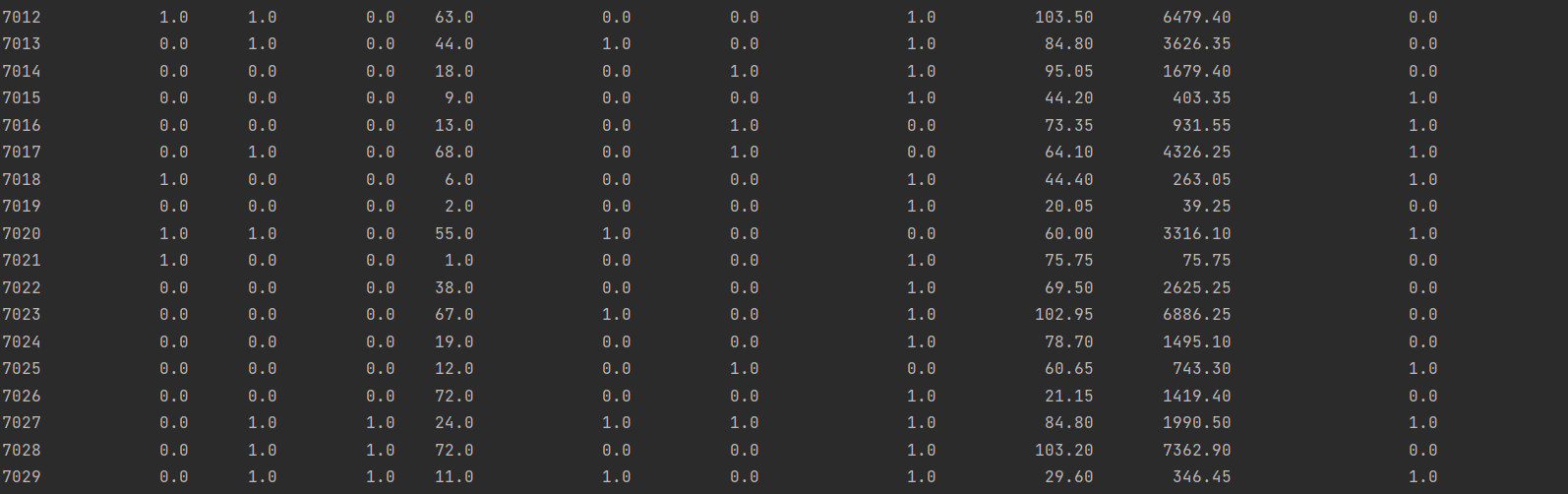
indices = np.argsort(scores)[::-1]

return list(feature\_data.columns.values[indices[0:model.k]])

fea\_name = [i for i in X.columns if i in SelectName(X,fs)]

X\_train = pd.DataFrame(X\_train\_fs,columns = fea\_name)

代码运行如下：



五．构建模型

# 模型建立和评估

# 首先使用分层抽样的方式将数据划分训练集和测试集。

# 重新划分

# 分层抽样

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_train, y, test\_size=0.2, random\_state=0, stratify=y)

# print(X\_train.shape, X\_test.shape, y\_train.shape, y\_test.shape)

X\_train = pd.DataFrame(X\_train)

X\_test = pd.DataFrame(X\_test)

y\_train = pd.DataFrame(y\_train)

y\_test = pd.DataFrame(y\_test)

#修正索引

for i in[X\_train, X\_test, y\_train, y\_test]:

i.index= range(i.shape[0])

# 保存标准化训练和测试数据

st= StandardScaler()

num\_scaled\_train=pd.DataFrame(st.fit\_transform(X\_train[num\_cols]),columns=num\_cols)

num\_scaled\_test=pd.DataFrame(st.transform(X\_test[num\_cols]),columns=num\_cols)

X\_train\_scaled= pd.concat([X\_train.drop(num\_cols, axis= 1), num\_scaled\_train], axis= 1)

X\_test\_scaled= pd.concat([X\_test.drop(num\_cols, axis= 1), num\_scaled\_test], axis= 1)

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\_)

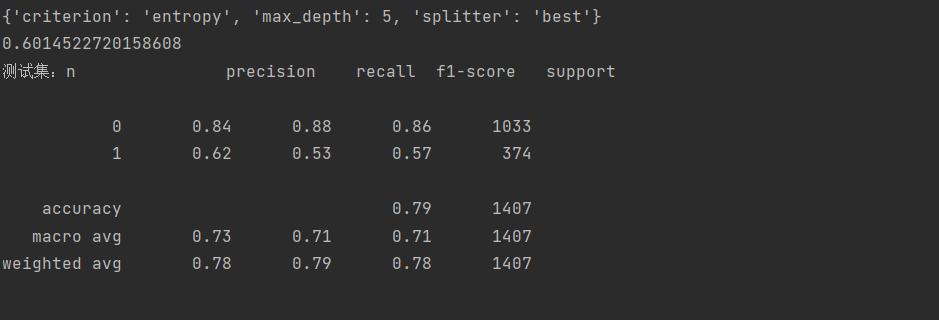
# print(GS.best\_score\_)

clf = GS.best\_estimator\_

test\_pred = clf.predict(X\_test)

print('测试集：n', classification\_report(y\_test, test\_pred))

代码运行如下：



六．评估模型

# 输出决策树属性重要性排序

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)

效果图如下：

