大数据分析技术综合设计

郭荣泓

目录

一、问题重述

在国内航空市场竞争日益激烈的背景下，A 航空公司在客户流失方面应该引起足够的重视。如何改善流失问题，继而提高客户满意度、忠诚度是航空公司维护自身市场并面对激烈竞争的一件大事，客户流失分析将成为帮助 A 航空公司开展持续改进活动的指南。

客户流失分析可以针对目前老客户进行分类预测。针对航空公司客户信息数据，可以进行老客户以及客户类型的定义（其中将飞行次数大于 6 次的客户定义为老客户，已流失客户定义为：第二年飞行次数与第一年飞行次数比例小于 50%的客户；准流失客户定义为：第二年飞行次数与第一年飞行次数比例在区间[50%,90%)内的客户；未流失客户定义为：第二年飞行次数与第一年飞行次数比例大于 90%的客户）。同时需要选取客户信息中的关键属性如：会员卡级别、客户类型（流失、准流失、未流失）、平均乘机时间间隔、平均折扣率、积分兑换次数、非乘机积分总和、单位里程票价、单位里程积分等。随机选取数据的 80%作为分类的训练样本，剩余的 20%作为测试样本。构建客户的流失模型，运用模型预测未来客户的类别归属。具体要求如下：

1）根据要求，对三类客户进行筛选和定义；

2）按照 8：2 的比例，随机划分训练数据和测试数据，并将训练数据和测试数据分别写入 mysql 数据库；

3）分别从 mysql 数据库中读取训练数据和测试数据，并进行标准化；

4）分别采用决策树，支持向量机，BP 神经网络三种方法预测未来客户的类别归属，对预测结果进行评估，并对三种方法的结果进行比较分析。

二、数据的读取和预处理

首先，我们要先对数据进行读取和处理，这一部分的数据操作大部分基于 python 的 pandas 库。根据题意，客户流失分析的对象主要是老客户，所以可以先将属于老客户的数据筛选出来，也就是用题干中的条件：飞行次数大于 6 次进行筛选。

```
## 读取数据
import os
import pandas as pd
os.chdir(r"C:/Users/Mac/Desktop")
oridata = pd.read_csv("air_data.csv")

## 筛选出老客户
data = oridata.loc[oridata['FLIGHT_COUNT']>6]
```

接着，我们对老客户进行类型的划分，划分的依据是“第二年飞行次数与第一年飞行次数比例 “。该比例大于 90% 的客户定义为未流失客户（类型 A)；比例在区间 [50%,90%) 内的客户定义为准流失客户（类型 B)；已流失客户定义为该比例小于 50% 的客户（类型 C)。

```
## 定义各类型客户
data['kind'] = data['L1Y_Flight_Count'] / data['P1Y_Flight_Count']
ind1 = data['kind']>=0.9
ind2 = (data['kind']>=0.5)&(data['kind']<0.9)
ind3 = data['kind']<0.5
data['kind'].loc[ind1] = "A"
data['kind'].loc[ind2] = "B"
data['kind'].loc[ind3] = "C"
```

接下来对其他变量做一些处理，或者生成一些新的变量。譬如利用总票价和里程数得到单位里程票价，利用总积分和里程数得到单位历程积分数。最后，只保留所有感兴趣的变量。观察数据可以发现有缺失值的记录占比极少，因此我们

选择直接删掉有缺失值的乘客记录。

```
## 仅保留需要的数据
data.rename(columns={'FFP_TIER':'tier', 'AVG_INTERVAL':'avggap',
                     'avg_discount':'discount', 'EXCHANGE_COUNT':'exchange',
                     'Point_NotFlight':'point'}, inplace=True)
data['avgprice'] = (data['SUM_YR_1']+data['SUM_YR_1'])/data['SEG_KM_SUM']
data['avgpoints'] = data['Points_Sum']/data['SEG_KM_SUM']
data = data[['tier','avggap','discount','exchange','point','avgprice','avgpoints','kind']]
data.dropna(inplace=True)
```

接下来按照题目要求，我们将已经初步处理完成的数据分割为训练集和测试集。分割按照 8：2 的比例，依据乘客类型分层进行。最终得到的是 26088 个训练样本，6523 个测试样本，共 7 个特征，三种目标类别。

```
## 分层划分为训练集和测试集
from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(test_size=0.2)
for ind1,ind2 in split.split(data, data['kind']):
    train = data.iloc[ind1]
    test = data.iloc[ind2]
```

将这些数据入库 MySQL，具体做法是使用 sqlalchemy 模块建立一个与数据库的连接，然后利用 pandas 的 to_sql 方法将 dataframe 结构中的数据直接导入数据库。（该方法默认无表则创建新表，有表则覆盖旧表）

```
## 将训练集和测试集分别导入 MYSQL
from sqlalchemy import create_engine
connect = create_engine('mysql+pymysql://root:6188@localhost:3306/MYSQL?charset=utf8')
train.to_sql('train',connect,if_exists='replace',index=False,chunksize=100)
test.to_sql('test',connect,if_exists='replace',index=False,chunksize=100)
```

数据成功入库后，本部分任务完成。

三、模型与评估

3.1 数据出库和标准化流程

这一部分将针对客户类型建立几个预测模型，并作出评估。首先，把第二部分存入数据库的数据提取出来，仍使用 pandas 库自带的 read_sql 函数来完成。观察数据，所有特征变量都是数值型的，于是为了便于之后的模型使用，我们对所有特征变量统一做标准化处理。对于目标变量，我们也对其进行编码，即使用 sklearn 的 OrdinalEncoder 方法将原来的字符串变量对应变为数字变量 1～3，以便于之后的建模。注意以上处理也同时对测试集进行。

```
## 从 MYSQL 读取数据
connect = create_engine('mysql+pymysql://root:6188@localhost:3306/MYSQL?charset=utf8')
trainset = pd.read_sql('train', connect)
testset = pd.read_sql('test', connect)
X_train = trainset.iloc[:, :7]
y_train = trainset['kind']
X_test = testset.iloc[:, :7]
y_test = testset['kind']

## 数据标准化
from sklearn.preprocessing import StandardScaler, OrdinalEncoder
scale = StandardScaler()
scale.fit(X_train)
X_train = scale.transform(X_train)
X_test = scale.transform(X_test)
encode = OrdinalEncoder(dtype=int)
encode.fit(y_train.values.reshape(-1,1))
y_train = encode.transform(y_train.values.reshape(-1,1))
y_test = encode.transform(y_test.values.reshape(-1,1))
```

3.2 支持向量机

下面进入建模阶段，首先我们使用支持向量机来构建一个分类器。支持向量机是一类按监督学习方式对数据进行二元分类的广义线性分类器，其决策边界是对样本求解的一个最大边距超平面。由于可以在 svm 基础上使用核函数技术，

因此该分类器也可以适用于非线性的分类问题。

先简单地试验线性 svm，得到的结果较差，因此我们选用高斯核函数（rbf）来对特征向量进行映射。至于具体参数 C 和 gamma，可以使用网格搜索技术，以十折检验的准确率为标准来找出最合适的一组参数。

```
## 网格搜索 SVM 的最佳参数，并得到对应的 cv 误判率
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import KFold, GridSearchCV
# 设置 cv 折数，定义参数表格
fold = KFold(n_splits=10, random_state=42)
param_range = {'C': [0.1, 1, 10, 100, 1000],
               'gamma': np.logspace(-1, 4, 6)}
# 进行网格搜索
svm_clf = GridSearchCV(SVC(kernel='rbf', max_iter=200), cv=fold,
                       param_grid=param_range)
svm_clf.fit(X_train, y_train)
# 得到所有组合的 cv 误判和最佳参数组合
svm_mcr = pd.DataFrame(
    (svm_clf.cv_results_['mean_test_score']).reshape(5, -1),
    columns=param_range['gamma'], index=param_range['C'])
print("高斯核 SVM 各参数组合的 cv 准确率表，行为 C 值，列为 gamma 值\n", svm_mcr)
```

注意到由于 SVM 的收敛速度较慢，为了提高效率这里设置了 max_iter，即最大迭代次数为 200 次，因此最终模型效果会有所下降（但因为速度的极大提升，这是值得的）。输出得到了如下的参数组合以及它们各自的模型效果：

```
高斯核SVM各参数组合的cv准确率表，行为C值，列为gamma值
          0.1       1.0       10.0      100.0     1000.0    10000.0
0.1   0.414673  0.561177  0.479730  0.500734  0.572255  0.287182
1.0   0.383358  0.572984  0.573942  0.538986  0.572256  0.287373
10.0  0.324208  0.493293  0.569496  0.572332  0.209368  0.304785
100.0 0.366179  0.531465  0.535881  0.571527  0.209368  0.304785
1000.0 0.382356  0.508243  0.505333  0.571527  0.209368  0.304785
```

从上面的结果可以看到准确率上表现最好的是C=10，gamma=1 这组参数，因此我们将选用这组参数来构建最终的 svm 分类器。该分类器在最终测试集上

的预测效果将在 3.5 展示。

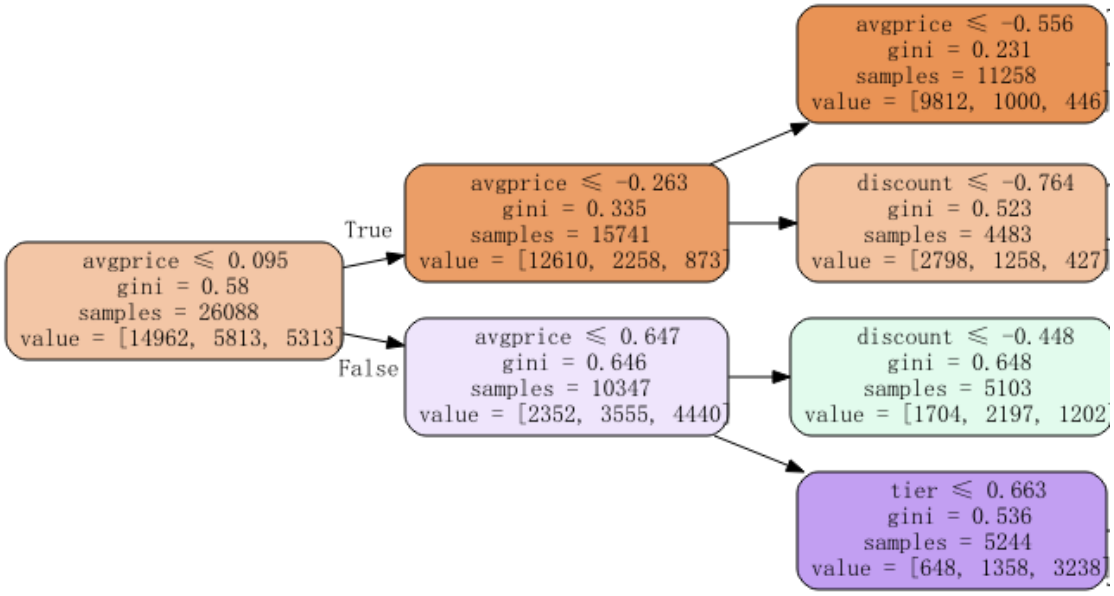
3. 3 决策树

接下来使用决策树来构建一个分类器。决策树也是一类监督学习方法，它试图在已知各种情况发生概率的基础上，通过构造树结构来评价项目风险，判断决策的可行性，是直观运用概率分析的一种图解法，因为这种决策分支画成图形很像一棵树的枝干而得名。

决策树的评判标准主要有两种，基尼系数和交叉熵。两者多数情况下差别不大，这里选用 gini 标准，即每一树的分叉均选择使得 gini 最小的特征和对应分界点。

```
## 决策树分类器
from sklearn.tree import DecisionTreeClassifier
tree_clf = DecisionTreeClassifier(criterion='gini', max_depth=20)
tree_clf.fit(X_train, y_train)
# 可视化
import graphviz
from sklearn.tree import export_graphviz
dot_data = export_graphviz(
    tree_clf, out_file=None, max_depth=3,
    feature_names=['tier','avggap','discount',
                   'exchange','point','avgprice','avgpoints'],
    filled=True, rounded=True,  special_characters=True)
graphviz.Source(dot_data)
```

由于样本较多，有必要对决策树做一些限制以避免过拟合，并加快算法速度，这里选择设置树的最大深度为 20。输出得到以下的决策树可视化结果（部分）：



决策树分类器在最终测试集上的预测效果将在 3.5 展示。

3.4 神经网络

下面我们将使用神经网络建模构造一个分类器。误差反向传播算法（BP），是一种解决多层神经网络隐含层连接权重学习问题的方案，具有任意复杂模式分类能力和优良的多维函数映射能力，解决了简单感知器不能解决的异或和一些其他问题。本部分将使用 tensorflow 框架来架设一个 BP 神经网络。

```
## BP 神经网络
import tensorflow.compat.v1 as tf
import numpy as np
tf.disable_v2_behavior()
tf.reset_default_graph()
# 开始架设神经网络
n_inputs = 1*7
n_hidden1 = 150
n_hidden2 = 150
n_outputs = 3
learning_rate = 0.1
n_epochs = 40
batch_size = 100
# 使用占位符节点来表示训练数据和目标
```

```
X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
y = tf.placeholder(tf.int32, shape=(None), name="y")
# 创建隐藏层和输出层
with tf.name_scope("dnn"):
    hidden1 = tf.layers.dense(X, n_hidden1, name="hidden1",
                              activation=tf.nn.elu)
    hidden2 = tf.layers.dense(hidden1, n_hidden2, name="hidden2",
                              activation=tf.nn.elu)
    logits = tf.layers.dense(hidden2, n_outputs, name="outputs")
    y_proba = tf.nn.softmax(logits)
```

以上搭建了一个含有两个中间层的神经网络，两个隐藏层均为 150 个神经元。激活函数使用 elu 函数，该函数的优点是平均输出接近于 0，有利于减轻梯度消失问题，虽然在计算量上大于一般 relu 函数，但收敛速度更快，在测试集上表现也更好。在模型的输出层，我们利用 softmax 函数来计算各类别的概率。

```
# 使用交叉熵计算代价函数
with tf.name_scope("loss"):
    # 根据通过 softmax 之前的输出和整数形式的标签计算交叉熵，输出一个交叉熵张量。
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(
        labels=y, logits=logits)
    # 计算交叉熵张量的平均数作为代价函数
    loss = tf.reduce_mean(xentropy, name="loss")
# 定义训练模式
with tf.name_scope("train"):
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    training_op = optimizer.minimize(loss)
# 评估神经网络
with tf.name_scope("eval"):
    # 将每个样本预测概率前 k 的标签与真实标签作对比，输出一个布尔张量
    correct = tf.nn.in_top_k(logits, y, k=1)
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
# 创建初始化变量节点
init = tf.global_variables_initializer()
# 创建保存节点
saver = tf.train.Saver()
```

以上代码设置了网络的更多细节，如使用交叉熵作为代价函数，GradientDescentOptimizer 作为优化器来执行梯度下降的 BP 算法。节点 eval

则给出了评估网络效果（准确率）的函数。接下来就可以开始训练网络了。

```
# 开始训练神经网络
def shuffle_batch(X, y, batch_size):
    rnd_idx = np.random.permutation(len(X))
    n_batches = len(X) // batch_size
    for batch_idx in np.array_split(rnd_idx, n_batches):
        X_batch, y_batch = X[batch_idx], y[batch_idx]
        yield X_batch, y_batch
with tf.Session() as sess:
    init.run()
    for epoch in range(n_epochs):
        # 随机梯度下降一次迭代
        for X_batch, y_batch in shuffle_batch(X_train, y_train.ravel(), batch_size):
            sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
        # 每次迭代进行一次评估
        acc_batch = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
        acc_val = accuracy.eval(feed_dict={X: X_test, y: y_test.ravel()})
        print(epoch, "Batch accuracy:", acc_batch, "Val accuracy:", acc_val)
    save_path = saver.save(sess, r'./Tensorflow Model\guo_model.ckpt')
```

训练中使用了一种特殊的技巧：批量梯度下降。这是一种重复随机生成小批次样本进行训练的方法，能极大地提高训练速度。打印每一批次训练过后的批次预测准确率和验证集准确率，得到以下结果（部分）：

```
30 Batch accuracy: 0.81 Val accuracy: 0.7574735
31 Batch accuracy: 0.76 Val accuracy: 0.756707
32 Batch accuracy: 0.81 Val accuracy: 0.7583934
33 Batch accuracy: 0.8 Val accuracy: 0.7560938
34 Batch accuracy: 0.8 Val accuracy: 0.756707
35 Batch accuracy: 0.79 Val accuracy: 0.7557872
36 Batch accuracy: 0.7 Val accuracy: 0.7553273
37 Batch accuracy: 0.78 Val accuracy: 0.7571669
38 Batch accuracy: 0.75 Val accuracy: 0.7565537
39 Batch accuracy: 0.74 Val accuracy: 0.7570136
```

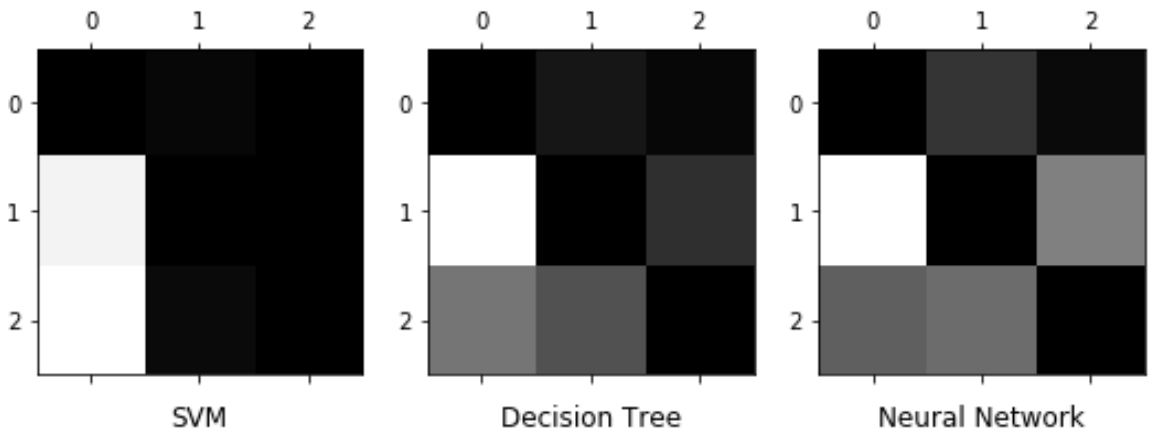
可以看到 BP 神经网络的效果偏差。

3.5 模型评估

这部分我们对已建好的模型进行效果评估，首先检查它们在训练集上的拟合

效果。按照各模型的拟合结果，统计每个类别下分别错判为其它两种类别的概率，列出一个 3×3 混淆矩阵，并据此绘制灰度图：

```
# %%
## 计算混淆矩阵
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
def confuse_mat(y_true, y_pred):
    confuse_mat = confusion_matrix(y_true, y_pred)
    # 比较错误率而不是错误数
    row_sums = confuse_mat.sum(axis=1, keepdims=True)
    norm_confuse_mat = confuse_mat / row_sums
    # 排除对角线的干扰
    np.fill_diagonal(norm_confuse_mat, 0)
    return norm_confuse_mat
## 绘制混淆图
fig, axes = plt.subplots(1, 3, figsize=(9,9))
axes[0].matshow(confuse_mat(y_train, svm_clf.predict(X_train)), cmap=plt.cm.gray)
axes[0].set_title("SVM", y=-0.2)
axes[1].matshow(confuse_mat(y_train, tree_clf.predict(X_train)), cmap=plt.cm.gray)
axes[1].set_title("Decision Tree", y=-0.2)
with tf.Session() as sess:
    saver.restore(sess, r'./Tensorflow Model\guo_model.ckpt')
    Z = logits.eval(feed_dict={X: X_train})
    y_pred = np.argmax(Z, axis=1)
axes[2].matshow(confuse_mat(y_train, y_pred), cmap=plt.cm.gray)
axes[2].set_title("Neural Network", y=-0.2)
```



上图展示了各类别下的误判概率，非对角线色块的色调越暗则占比越小。可以大致上看出 SVM 在类别 1（准流失）和类别 2（已流失）上的判断表现不错，但在类别 0（未流失）上有严重的误判；决策树与 SVM 的情况类似，但在类别 0 上表现稍好；BP 神经网络在各个类别上均有些许误判。

接下来分别计算各分类器在训练集和测试集上的准确率，以此来观察各分类器的表现：

可以看到 SVM 在训练集测试集上准确率均较低，显示出欠拟合的特征；决策树在训练集上的表现相当出色，但在测试集上表现不佳，显示出过拟合的特征；BP 神经网络在训练集和测试集上均是中规中矩，体现出了一定的拟合能力和泛化能力。

四、总结

本报告从客户流失分析的角度出发，利用航空公司客户信息数据对客户类型进行了定义和筛选，同时选取客户信息中的关键属性如：会员卡级别、客户类型、

平均乘机时间间隔、平均折扣率、积分兑换次数、非乘机积分总和、单位里程票价、单位里程积分等构建客户流失模型，并运用这些模型来预测未来客户的类别归属。

在本文运用的三个模型中，SVM 显然是难以适用于该场景的，具体体现在其对训练集的拟合能力有限，展现出了明显的欠拟合特征。并且由于算法速度过慢，该模型也不适用于本场景下的较大样本数据。决策树则展现出了其优越的拟合能力，对训练集的预测准确率达到 95%，但在测试集上则表现不佳，没有体现出可用的泛化能力，展现出了过拟合的特征，因而也需要进一步改进，如改用随机树方法，或使用随机森林算法。BP 神经网络则在训练集和测试集上均达到80%左右的准确率，体现出了一定的泛化能力，可以说是三个模型中总体表现最好的。如果需要进一步改进模型，可以从增加神经元或隐藏层，更换梯度下降算法等角度入手，继续提高在训练集上的表现。