**大数据分析技术综合设计**

**郭荣泓**

**目录**

[**一、问题重述** 3](#_Toc60350531)

[**二、数据的读取和预处理** 4](#_Toc60350532)

[**三、模型与评估** 6](#_Toc60350533)

[**3.1数据出库和标准化流程** 6](#_Toc60350534)

[**3.2支持向量机** 6](#_Toc60350535)

[**3.3决策树** 8](#_Toc60350536)

[**3.4神经网络** 9](#_Toc60350537)

[**3.5模型评估** 11](#_Toc60350538)

[**四、总结** 13](#_Toc60350539)

**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。

1. ## 读取数据
2. **import** os
3. **import** pandas as pd
4. os.chdir(r"C:/Users/Mac/Desktop")
5. oridata = pd.read\_csv("air\_data.csv")
7. ## 筛选出老客户
8. 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）。

1. ## 定义各类型客户
2. data['kind'] = data['L1Y\_Flight\_Count'] / data['P1Y\_Flight\_Count']
3. ind1 = data['kind']>=0.9
4. ind2 = (data['kind']>=0.5)&(data['kind']<0.9)
5. ind3 = data['kind']<0.5
6. data['kind'].loc[ind1] = "A"
7. data['kind'].loc[ind2] = "B"
8. 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。

1. ## 仅保留需要的数据
2. data.rename(columns={'FFP\_TIER':'tier', 'AVG\_INTERVAL':'avggap',
3. 'avg\_discount':'discount', 'EXCHANGE\_COUNT':'exchange',
4. 'Point\_NotFlight':'point'}, inplace=True)
5. data['avgprice'] = (data['SUM\_YR\_1']+data['SUM\_YR\_1'])/data['SEG\_KM\_SUM']
6. data['avgpoints'] = data['Points\_Sum']/data['SEG\_KM\_SUM']
7. data = data[['tier','avggap','discount','exchange','point','avgprice','avgpoints','kind']]
8. 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。

1. ## 分层划分为训练集和测试集
2. **from** sklearn.model\_selection **import** StratifiedShuffleSplit
3. split = StratifiedShuffleSplit(test\_size=0.2)
4. **for** ind1,ind2 **in** split.split(data, data['kind']):
5. train = data.iloc[ind1]
6. 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）

1. ## 将训练集和测试集分别导入MYSQL
2. **from** sqlalchemy **import** create\_engine
3. connect = create\_engine('mysql+pymysql://root:6188@localhost:3306/MYSQL?charset=utf8')
4. train.to\_sql('train',connect,if\_exists='replace',index=False,chunksize=100)
5. 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。

1. ## 从MYSQL读取数据
2. connect = create\_engine('mysql+pymysql://root:6188@localhost:3306/MYSQL?charset=utf8')
3. trainset = pd.read\_sql('train', connect)
4. testset = pd.read\_sql('test', connect)
5. X\_train = trainset.iloc[:, :7]
6. y\_train = trainset['kind']
7. X\_test = testset.iloc[:, :7]
8. y\_test = testset['kind']
10. ## 数据标准化
11. **from** sklearn.preprocessing **import** StandardScaler, OrdinalEncoder
12. scale = StandardScaler()
13. scale.fit(X\_train)
14. X\_train = scale.transform(X\_train)
15. X\_test = scale.transform(X\_test)
16. encode = OrdinalEncoder(dtype=int)
17. encode.fit(y\_train.values.reshape(-1,1))
18. y\_train = encode.transform(y\_train.values.reshape(-1,1))
19. y\_test = encode.transform(y\_test.values.reshape(-1,1))

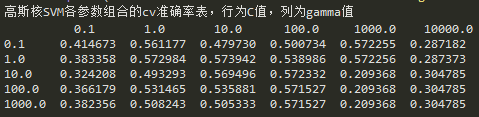
**3.2支持向量机**

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](https://baike.baidu.com/item/%E7%9B%91%E7%9D%A3%E5%AD%A6%E4%B9%A0/9820109)方式对数据进行[二元分类](https://baike.baidu.com/item/%E4%BA%8C%E5%85%83%E5%88%86%E7%B1%BB/15635322)的广义线性分类器，其[决策边界](https://baike.baidu.com/item/%E5%86%B3%E7%AD%96%E8%BE%B9%E7%95%8C/22778546)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。

1. ## 网格搜索SVM的最佳参数，并得到对应的cv误判率
2. **import** numpy as np
3. **from** sklearn.svm **import** SVC
4. **from** sklearn.model\_selection **import** KFold, GridSearchCV
5. # 设置cv折数，定义参数表格
6. fold = KFold(n\_splits=10, random\_state=42)
7. param\_range = {'C': [0.1, 1, 10, 100, 1000],
8. 'gamma': np.logspace(-1, 4, 6)}
9. # 进行网格搜索
10. svm\_clf = GridSearchCV(SVC(kernel='rbf', max\_iter=200), cv=fold,
11. param\_grid=param\_range)
12. svm\_clf.fit(X\_train, y\_train)
13. # 得到所有组合的cv误判和最佳参数组合
14. svm\_mcr = pd.DataFrame(
15. (svm\_clf.cv\_results\_['mean\_test\_score']).reshape(5, -1),
16. columns=param\_range['gamma'], index=param\_range['C'])
17. **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：



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。

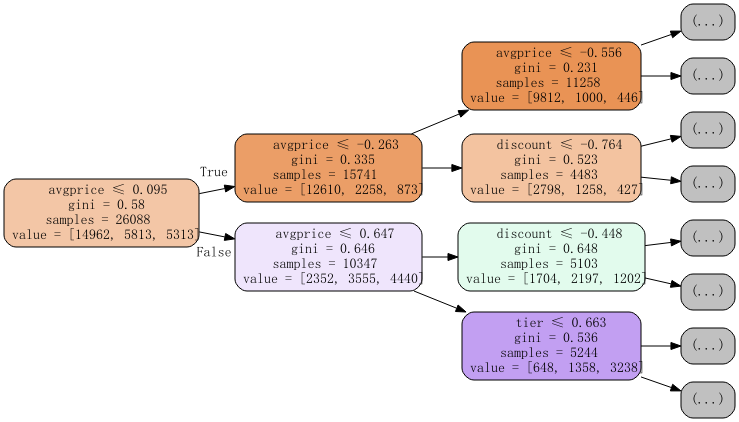
**3.3决策树**

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](https://baike.baidu.com/item/%E5%9F%BA%E7%A1%80/32794)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。

1. ## 决策树分类器
2. **from** sklearn.tree **import** DecisionTreeClassifier
3. tree\_clf = DecisionTreeClassifier(criterion='gini', max\_depth=20)
4. tree\_clf.fit(X\_train, y\_train)
5. # 可视化
6. **import** graphviz
7. **from** sklearn.tree **import** export\_graphviz
8. dot\_data = export\_graphviz(
9. tree\_clf, out\_file=None, max\_depth=3,
10. feature\_names=['tier','avggap','discount',
11. 'exchange','point','avgprice','avgpoints'],
12. filled=True, rounded=True,  special\_characters=True)
13. 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](https://baike.baidu.com/item/%E8%AF%AF%E5%B7%AE%E5%8F%8D%E5%90%91%E4%BC%A0%E6%92%AD%E7%AE%97%E6%B3%95)（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。

1. ## BP神经网络
2. **import** tensorflow.compat.v1 as tf
3. **import** numpy as np
4. tf.disable\_v2\_behavior()
5. tf.reset\_default\_graph()
6. # 开始架设神经网络
7. n\_inputs = 1\*7
8. n\_hidden1 = 150
9. n\_hidden2 = 150
10. n\_outputs = 3
11. learning\_rate = 0.1
12. n\_epochs = 40
13. batch\_size = 100
14. # 使用占位符节点来表示训练数据和目标
15. X = tf.placeholder(tf.float32, shape=(None, n\_inputs), name="X")
16. y = tf.placeholder(tf.int32, shape=(None), name="y")
17. # 创建隐藏层和输出层
18. with tf.name\_scope("dnn"):
19. hidden1 = tf.layers.dense(X, n\_hidden1, name="hidden1",
20. activation=tf.nn.elu)
21. hidden2 = tf.layers.dense(hidden1, n\_hidden2, name="hidden2",
22. activation=tf.nn.elu)
23. logits = tf.layers.dense(hidden2, n\_outputs, name="outputs")
24. y\_proba = tf.nn.softmax(logits)

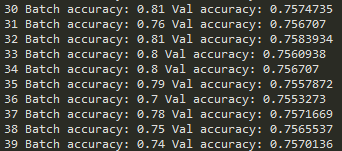
以上搭建了一个含有两个中间层的神经网络，两个隐藏层均为150个神经元。激活函数使用elu函数，该函数的优点是平均输出接近于0，有利于减轻梯度消失问题，虽然在计算量上大于一般relu函数，但收敛速度更快，在测试集上表现也更好。在模型的输出层，我们利用softmax函数来计算各类别的概率。

1. # 使用交叉熵计算代价函数
2. with tf.name\_scope("loss"):
3. # 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。
4. xentropy = tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits(
5. labels=y, logits=logits)
6. # 计算交叉熵张量的平均数作为代价函数
7. loss = tf.reduce\_mean(xentropy, name="loss")
8. # 定义训练模式
9. with tf.name\_scope("train"):
10. optimizer = tf.train.GradientDescentOptimizer(learning\_rate)
11. training\_op = optimizer.minimize(loss)
12. # 评估神经网络
13. with tf.name\_scope("eval"):
14. # 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
15. correct = tf.nn.in\_top\_k(logits, y, k=1)
16. accuracy = tf.reduce\_mean(tf.cast(correct, tf.float32))
17. # 创建初始化变量节点
18. init = tf.global\_variables\_initializer()
19. # 创建保存节点
20. 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。

1. # 开始训练神经网络
2. **def** shuffle\_batch(X, y, batch\_size):
3. rnd\_idx = np.random.permutation(len(X))
4. n\_batches = len(X) // batch\_size
5. **for** batch\_idx **in** np.array\_split(rnd\_idx, n\_batches):
6. X\_batch, y\_batch = X[batch\_idx], y[batch\_idx]
7. **yield** X\_batch, y\_batch
8. with tf.Session() as sess:
9. init.run()
10. **for** epoch **in** range(n\_epochs):
11. # 随机梯度下降一次迭代
12. **for** X\_batch, y\_batch **in** shuffle\_batch(X\_train, y\_train.ravel(), batch\_size):
13. sess.run(training\_op, feed\_dict={X: X\_batch, y: y\_batch})
14. # 每次迭代进行一次评估
15. acc\_batch = accuracy.eval(feed\_dict={X: X\_batch, y: y\_batch})
16. acc\_val = accuracy.eval(feed\_dict={X: X\_test, y: y\_test.ravel()})
17. **print**(epoch, "Batch accuracy:", acc\_batch, "Val accuracy:", acc\_val)
18. 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）：

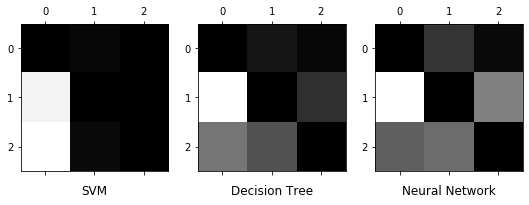


可以看到BP神经网络的效果偏差。

**3.5模型评估**

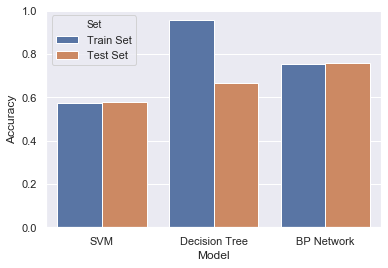
这部分我们对已建好的模型进行效果评估，首先检查它们在训练集上的拟合效果。按照各模型的拟合结果，统计每个类别下分别错判为其它两种类别的概率，列出一个3×3混淆矩阵，并据此绘制灰度图：

1. # %%
2. ## 计算混淆矩阵
3. **from** sklearn.metrics **import** confusion\_matrix
4. **import** matplotlib.pyplot as plt
5. **def** confuse\_mat(y\_true, y\_pred):
6. confuse\_mat = confusion\_matrix(y\_true, y\_pred)
7. # 比较错误率而不是错误数
8. row\_sums = confuse\_mat.sum(axis=1, keepdims=True)
9. norm\_confuse\_mat = confuse\_mat / row\_sums
10. # 排除对角线的干扰
11. np.fill\_diagonal(norm\_confuse\_mat, 0)
12. **return** norm\_confuse\_mat
13. ## 绘制混淆图
14. fig, axes = plt.subplots(1, 3, figsize=(9,9))
15. axes[0].matshow(confuse\_mat(y\_train, svm\_clf.predict(X\_train)), cmap=plt.cm.gray)
16. axes[0].set\_title("SVM", y=-0.2)
17. axes[1].matshow(confuse\_mat(y\_train, tree\_clf.predict(X\_train)), cmap=plt.cm.gray)
18. axes[1].set\_title("Decision Tree", y=-0.2)
19. with tf.Session() as sess:
20. saver.restore(sess, r'./Tensorflow Model\guo\_model.ckpt')
21. Z = logits.eval(feed\_dict={X: X\_train})
22. y\_pred = np.argmax(Z, axis=1)
23. axes[2].matshow(confuse\_mat(y\_train, y\_pred), cmap=plt.cm.gray)
24. 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。