南京信息工程大学

数据挖掘与安全课程设计

题　　目　　酒店退订预测

学生姓名　蒋谊

学　　号　20178314040

学　　院　计算机与软件学院

专　　业　信息安全

指导教师　闫雷鸣

二〇二零年六月八日

目 录

酒店退订预测

蒋谊

南京信息工程大学计算机与软件学院，江苏 南京 210044

摘要：酒店行业常常会面临订单取消的问题。为减少因消费者退订带来的损失，酒店可以通过数据分析对潜在的退订客户进行预测并提前联系，从而优化房间安排方案。本项目通过对2014年11月至2017年9月间收集到的119390条酒店用户信息数据进行预处理及可视化，并使用五种机器学习算法分别构建预测模型。此外，本项目还通过特征选择、集成学习等方式对模型进行优化。经优化后，预测模型的准确率达到了 0.975，f1-score达到了0.966。

关键词：酒店退订预测；机器学习；数据挖掘

Prediction of Hotel Cancellation

Jiang Yi

School of Computer and Software, NUIST, Nanjing 210044, China

Abstract：The hotel industry often faces the problem of cancellation. In order to reduce the losses caused by cancellation, hotels can use data analysis to predict potential unsubscribe customers and contact them in advance to optimize the room arrangement plan. This project preprocesses and visualizes 119,390 pieces of hotel user information data collected from November 2014 to September 2017, and uses five machine learning algorithms to build prediction models. In addition, the project also optimizes the model through feature selection and ensemble learning to help the hotel predict cancellation more accurately and reduce economic losses. After improvement, the accuracy achieved 0.975, the f1-score achieved 0.966.

Key words：Prediction of hotel cancellation；Machine Learning；Data mining

1 背景简介

酒店行业每天都要接待大批顾客，海量的相关性数据也由此相伴而生。每位顾客从开始选择酒店、预订客房，到入住，再到期满退房，这一系列的行为活动都包含了大量的数据信息。而酒店在获取这些数据信息后，可以充分挖掘出蕴含在其中的巨大商业价值，对顾客的各项行为数据进行预测分析，并据此来准确判断顾客的需求，帮助酒店更好地经营发展。

酒店退订，作为酒店销量预测中的不可控行为，常常困扰着酒店经营者。为减少因消费者退订带来的损失，酒店可以通过数据分析对潜在的退订客户进行预测，并根据不同特征的用户设置不同的退订规则，从而优化房间安排方案，将酒店退订带来的损失最小化[1]。本项目通过对 Hotel booking demand 数据集（即 hotel_bookings.csv 文件）进行分析，实现了对顾客的取消订单行为的预测，具体工作如下：

- 基于客户来源、客流分布时间、客户需求、先前退定记录等特征，对数据集进行探索性数据分析；
- 分别采用随机森林、决策树、AdaBoost、逻辑回归、人工神经网络算法，对预处理后的数据构建预测模型，并通过对比准确率和 F1-score、绘制 ROC 曲线等方式评价模型。
- 通过剔除无关字段、采用投票等方式，对预测模型进行优化。

2 数据集信息

2.1 数据来源

Hotel booking demand 数据集[2]是由葡萄牙学者 Nuno Antonio、Ana de Almeida 和 Luis Nunes 于 2018 年 10 月通过提取酒店物业管理系统 SQL 数据库收集，并在 2020 年 2 月由 Thomas Mock 和 Antoine Bichat 等人进行了数据清洗[3]。

本数据集描述了两家类型不同但结构相似的酒店在 2015 年 7 月 1 日到 2017 年 8 月 31 日间共 119390 条用户订单信息。这两家酒店中，一家是度假酒店，标记为"H1"，共有 40060 条数据；另一家酒店为城市酒店，标记为"H2"，共有 79330 条数据。每一条数据都代表了一条订单信息。此外，由于这些数据都是真实的酒店预定记录，因此数据集中包含的酒店和客户隐私信息已都被删除。

2.2 数据集字段描述

本数据集共包含 32 种特征，其中 is_cancel 为分类标签（0 为未退订，1 为退订），其余特征用于构建退订预测模型。具体特征名称、数据类型与描述如表 1 所示。

表 1 数据集字段描述

变量名	数据类型	变量描述
hotel	object	酒店种类（包括”H1”和”H2”）
is_canceled	int64	是否退订（0 为未退订，1 为退订）
lead_time	int64	客户提前预定的天数

arrival_date_year	int64	入住的年份
arrival_date_month	object	入住的月份
arrival_date_week_number	int64	入住的周份
arrival_date_day_of_month	int64	入住的日份
stays_in_weekend_nights	int64	（非工作日）顾客在酒店的天数
stays_in_week_nights	int64	（工作日）顾客在酒店的天数
adults	int64	成年人数
children	int64	儿童数量
babies	int64	宝宝数量
meal	object	预定的伙食种类
country	object	客户国籍简称
market_segment	object	细分市场
distribution_channel	object	预定渠道
is_repeated_guest	int64	先前是否预订过（0 为未订过，1 为预订过）
previous_cancellations	int64	该用户先前退订的次数
previous_bookings_not_canceled	int64	该用户先前未退订的次数
reserved_room_type	object	预留的房间类型编号
assigned_room_type	object	指定的房间类型编号
booking_changes	int64	预定变更次数
deposit_type	object	押金种类
agent	object	代理人的 ID
company	object	公司 ID
days_in_waiting_list	int64	在等待名单上的天数
customer_type	object	顾客种类
adr	float64	日均房价
required_car_parking_spaces	int64	需要的停车位数量
total_of_special_requests	int64	顾客所需的特殊需求数
reservation_status	object	最后的预定状态
reservation_status_date	object	最后预定状态的设置日期

3 探索性数据分析

探索性数据分析（Exploratory Data Analysis，EDA），是指对已有的数据，在尽量少的先验假定下进行探索通过作图、制表、方程拟合、计算特征量等手段探索数据的结构和规律的一种数据分析方法。接下来，我将对原始数据进行可视化分析和预处理操作。

3.1 缺失值处理

首先，载入数据，查看含有缺失值的属性，如图 1 所示。发现’children’、’country’、’agent’和’company’存在缺失值。

```
# 查看缺失值
t = data.isnull().sum()
print("NaN in columns (if any):")
t[t>0]
```

```
NaN in columns (if any):

children         4
country        488
agent        16340
company     112593
dtype: int64
```

图 1 缺失值查看

然后，将’country’属性替换为’Unknown’，其余属性缺失值填补为 0。此外，根据数据集原文档的介绍，’meal’属性值’Undefined’等价于’SC’，因此也将其替换。然后，存在部分行出现“僵尸订单”，即成人、儿童和婴儿都为 0。将这些“僵尸订单”去除后，打印数据集大小，如图 2 所示。

```
# 替换缺失值
nan_replacements = {"children:": 0.0,"country": "Unknown", "agent": 0, "company": 0}
data_cln = data.fillna(nan_replacements)

# 将"meal"中的"Undefined"替换为SC.
data_cln["meal"].replace("Undefined", "SC", inplace=True)

# 去除一些僵尸行(0 adults, 0 children and 0 babies)
zero_guests = list(data_cln.loc[data_cln["adults"]
                   + data_cln["children"]
                   + data_cln["babies"]==0].index)
data_cln.drop(data_cln.index[zero_guests], inplace=True)
data_cln.shape
```

```
(119210, 32)
```

图 2 缺失值处理过程

由图 2 可知，处理后的数据集剩下 119210 行，有 180 条“僵尸订单”被删除。

3.2 酒店订单基本信息分析

图 3 顾客国籍分布

首先，查看顾客的地域分布。先对每个国家的订单总数进行统计，再调用 plotly.express 包中的 choropleth 方法，可以将各个国家的订单数目可视化。可视化结果如图 3 所示。可以

看到，由于该数据集酒店来源于葡萄牙，因此顾客多分布于葡萄牙本国及其西欧邻国。其余顾客零散分布在世界各地。有趣的是，还有 2 位顾客来自于南极，推测可能是国家填写时出现了错误。

接着，分析房间种类与对应的价格。通过将 adr 数据除以每一单中成人和儿童的总数，得到不同房间的人均消费，并使用箱型图可视化，结果如图 4 所示。可以发现，大部分房间类型的每晚人均中位数都在 40 到 60 欧元区间内。由于房型设置原因，城市酒店不提供 H 和 L 型房间。

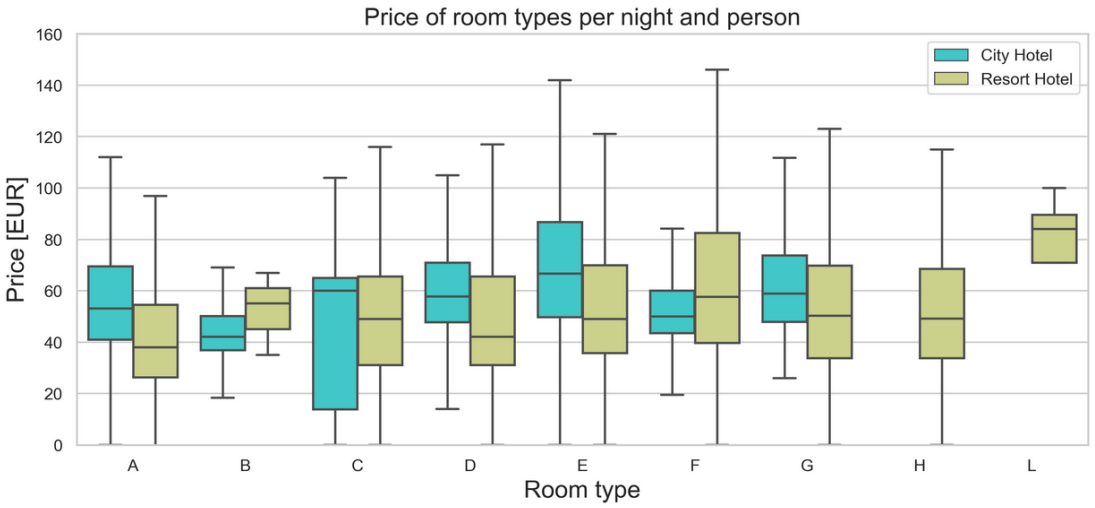

图 4 不同房间类型的每晚人均消费

然后，查看每人每晚开销再不同月份的波动情况。基于图 4 中不同房间的人均消费，按照 arrival_date_month 进行排序，并计算两种酒店价格的方差，绘制折线图，如图 5 所示。可以发现，度假酒店的每人每晚价格在八月达到顶峰，而冬季则处于淡季。相对于度假酒店的淡旺季分明，城市酒店的价格则相对稳定，大约在 60 欧元左右波动。

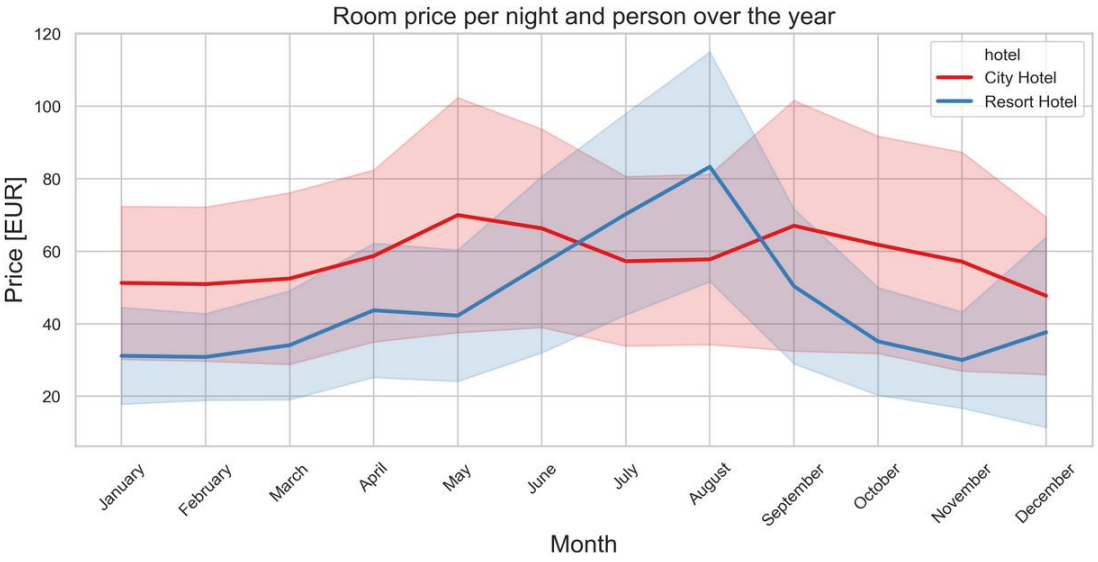

图 5 酒店每晚人均消费随时间变化图

最后，分析顾客预定酒店的方式。统计 market_segment 字段各种值的总数，并绘制饼图，如图 6 所示。可以看到，有近半数的订单源自在线预定。而组团或公司预定的订单总数仅占到总订单数的 20%。

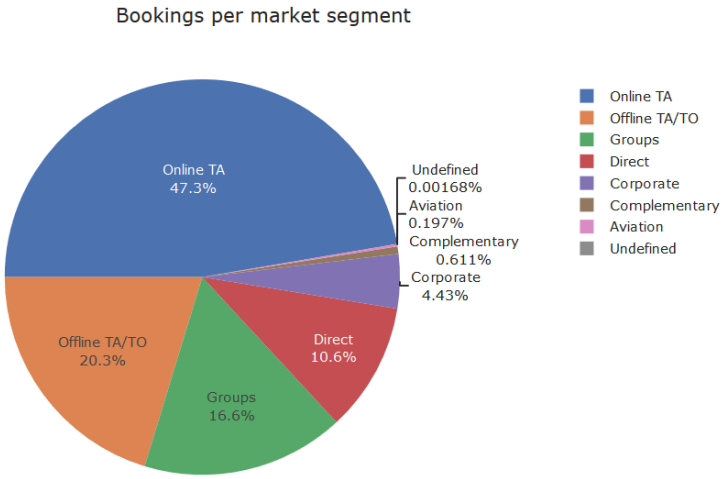

图 6 订单来源分布图

3.3 用户退订因素分析

为更好地进行酒店退订预测，我还对部分特征与退订结果间的变化关系进行可视化。首先，查看城市酒店和度假酒店的总退订数，如图 7 所示。在该数据集中，酒店总退订数为 44179 单，占总订单数的 37%，其中度假酒店共 11120 单退订，占其比重的 28%；而城市酒店退订数达到了 33079，占到了其总订单记录的 42%。

```
# 计算退订数及退订率
total_cancelations = data_cln["is_canceled"].sum()
rh_cancelations = data_cln.loc[data_cln["hotel"] == "Resort Hotel"]["is_canceled"].sum()
ch_cancelations = data_cln.loc[data_cln["hotel"] == "City Hotel"]["is_canceled"].sum()
rel_cancel = total_cancelations / data_cln.shape[0] * 100
rh_rel_cancel = rh_cancelations / data_cln.loc[data_cln["hotel"] == "Resort Hotel"].shape[0] * 100
ch_rel_cancel = ch_cancelations / data_cln.loc[data_cln["hotel"] == "City Hotel"].shape[0] * 100
print(f"Total bookings canceled: {total_cancelations:,} ({rel_cancel:.0f} %)")
print(f"Resort hotel bookings canceled: {rh_cancelations:,} ({rh_rel_cancel:.0f} %)")
print(f"City hotel bookings canceled: {ch_cancelations:,} ({ch_rel_cancel:.0f} %)")
```

```
Total bookings canceled: 44,199 (37 %)
Resort hotel bookings canceled: 11,120 (28 %)
City hotel bookings canceled: 33,079 (42 %)
```

图 7 计算退订数与退订率

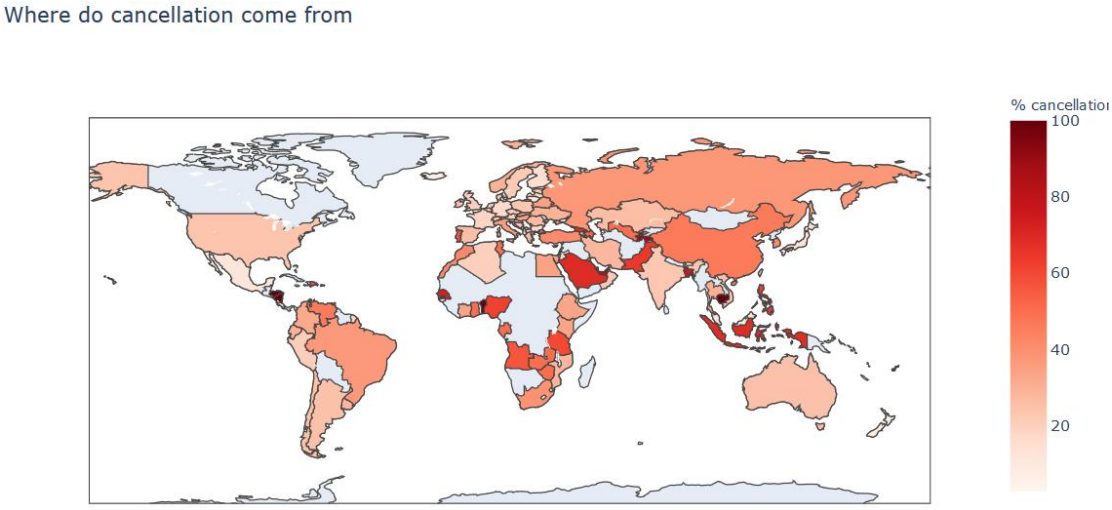

图 8 查看各个国家的退订率

接着，查看不同国家的用户退订率。统计每个国家的总订单数与它们'is_cancel'字段值为 1 的总数，得到每个国家的退订率，如图 8 所示。可以看到，大部分发达国家由于总订单数较多，因而退订比例基本相差不大，大约在 20%到 40%之间。而一些欠发达国家（非洲、东南亚等）由于总订单数极少，因而退订数对其比例的影响非常大。

然后，查看提前预定时间、日均房价、押金种类、订单来源、预定日期等因素与退订结果的影响，如图 9 所示。经分析，可以进行以下推测：

① 两类酒店退订者的提前预定时间显著超过按时入住者的提前预定时间，说明提前预定时间有可能与退订率有较为密切的联系。

② 对于退订的用户，其中三分之一最终支付的费用超过了正常入住的费用。在这种情况下，酒店并未因为顾客退订而受到损失。

③ 从订单来源的角度分析，组队订购酒店者的退订总数超过了其正常入住的总数，说明组队订购酒店者退订的可能性更高。

④ 城市酒店退订率全年都较为稳定，而度假酒店在夏季退订率最高，在春季和秋季逐渐下降，到冬季时退订率达到低谷。

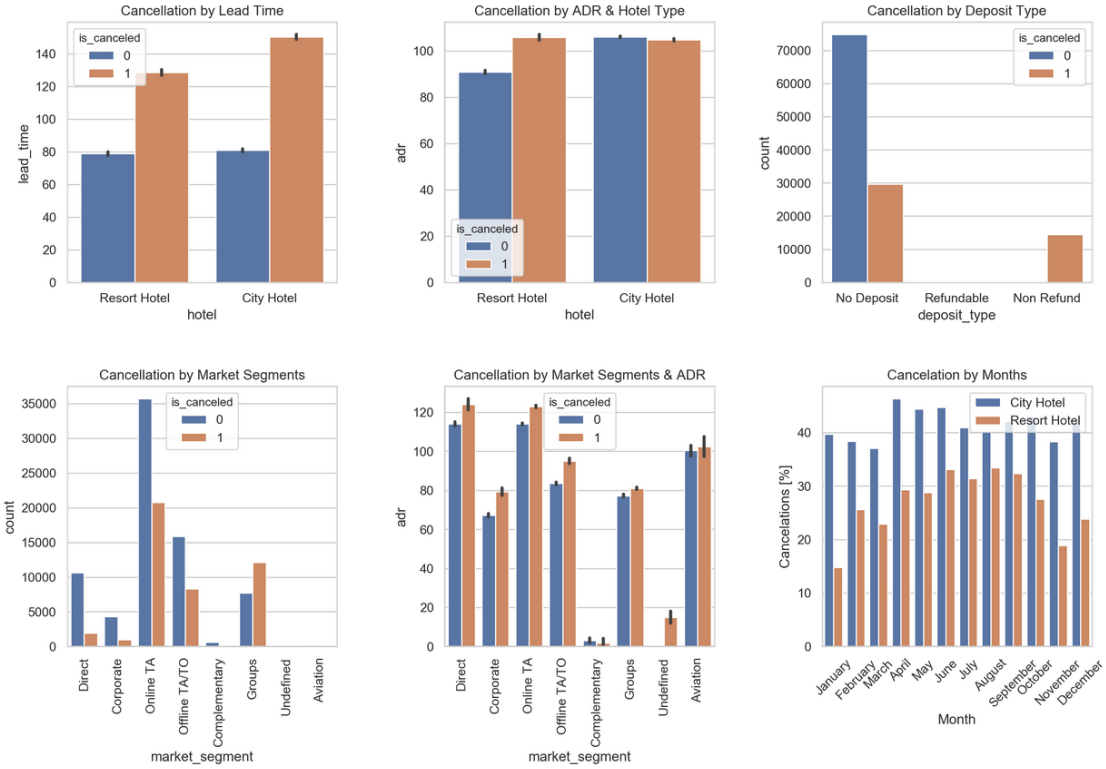

图 9 多种特征与退订情况的关系图

随后，再通过调用相关系数矩阵方法，计算各个特征对于退订结果的影响权重，并进行降序排列与可视化。影响权重前排名 20 的特征及其权重大小如图 10 所示。可以看到，提前预定时间对退订结果有最大的影响，特殊需求、是否需要停车位、客户预定是否变更、曾经是否有退订记录对于退订结果的影响也较大，都超过了 0.1。

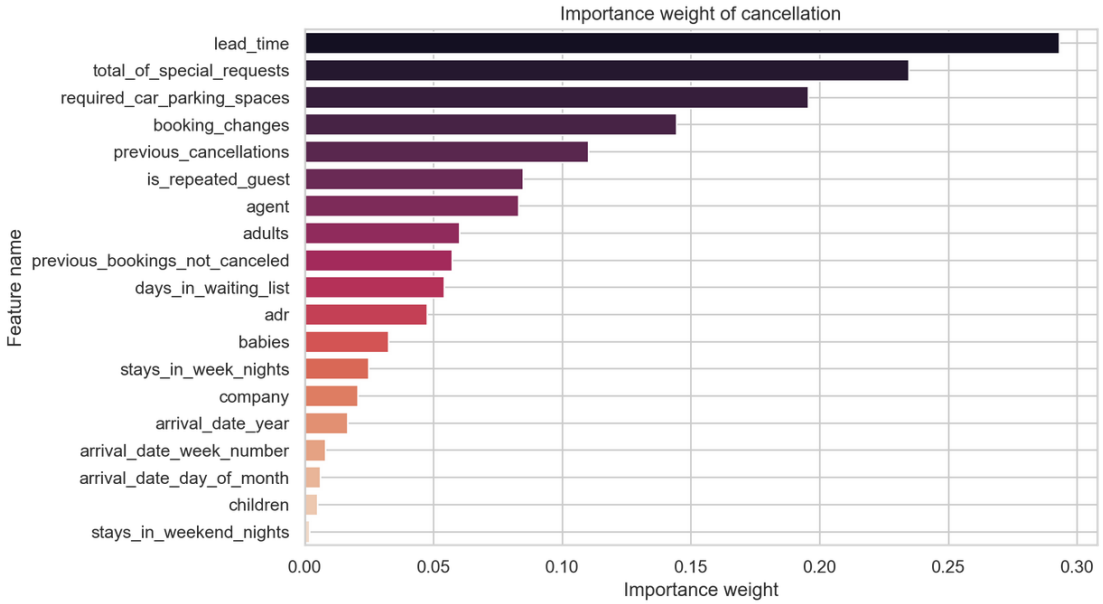

图 10 退订结果影响因素权重图

3.4 数据规范化与降维

数据的规范化的作用主要有两个：去掉量纲，使得指标之间具有可比性；将数据限制到一定区间，使得运算更为便捷。其中有两种规范化方法最为常见，分别是标准归一化和最大最小归一化。标准归一化基于原始数据的均值（mean）和标准差（standard deviation）进行数据的标准化，而最大最小归一化使用的是数据集中的最大值和最小值，来对原始数据的线性变换。

数据降维，作为预处理的一部分，可以有效降低数据为低，便于计算和可视化，并有助于有效信息的提取综合及无用信息的摈弃。常见的降维算法有主成分分析（PCA）和流形学习降维算法 Isomap。

接下来，对数据进行处理。由于数据集较大，降维算法运行慢，因此，我随机挑选 10%的数据进行降维可视化。首先，提取出数据集中的字符串类型的数据，采用 LabelEncoder()和 OneHotEncoder()编码方式对其进行处理，使其转化为数字类型的数据。接着，分别采用标准归一化和最大最小归一化对数据集进行标准化。将标准化后的数据分别采用 PCA 算法进行降维，降至两维，并进行训练。结果分别如图 11 所示。同样地，两种方法标准化后的数据分别用 Isomap 算法降维后结果如图 12 所示。其中蓝色代表退订数据，红色代表正常数据）

相比较而言，使用标准归一化的处理并降维后，退订数据相对集中，而使用最大最小归一化处理后，二者分布情况都较为分散。此外可以看到，由于两种方法的数据重合度都较高，但是 Isomap 算法能将一部分退订数据分离出来，因此其将为效果比 PCA 略好。

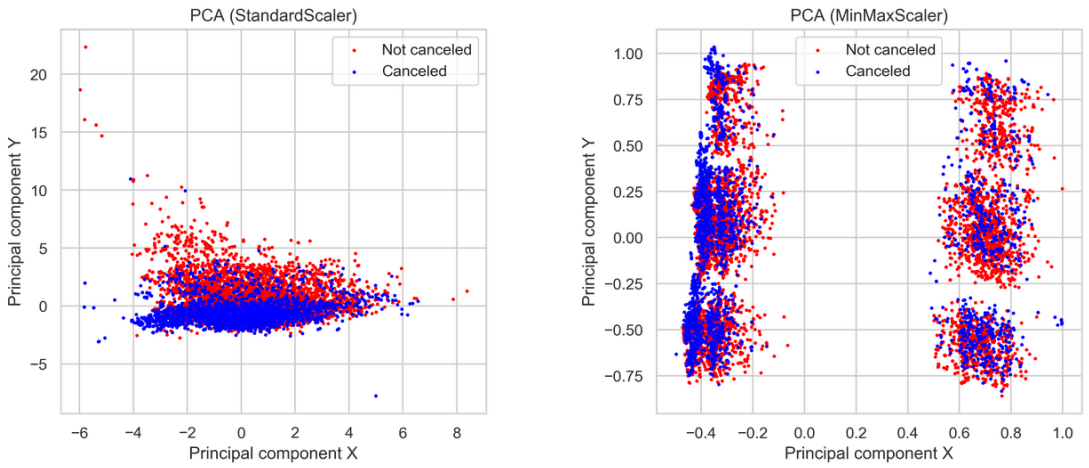

图 11 使用两种归一化方法和 PCA 降维后的效果图

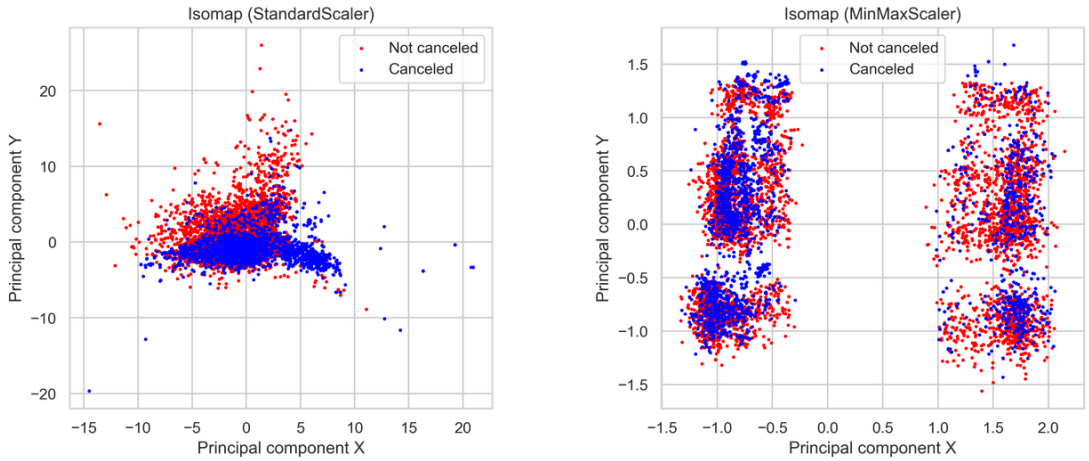

图 12 使用两种归一化方法和 Isomap 降维后的效果图

4 调参建模与模型评价

本节将介绍随机森林、决策树、AdaBoost、逻辑回归、人工神经网络等五种常用的二分类算法。我使用了 train_test_split 方法，将数据随机分为训练样本与测试样本，并设置测试样本比例为 0.25。为防止过拟合，我通过对人工神经网络分类器添加 dropout 和 early stopping，对其余机器学习算法采用五折交叉验证，从而构建酒店退订预测模型。本节的所有实验均采用 3.4 节中由 LabelEncoder()和 OneHotEncoder()编码后，经由 StandardScaler 标准化处理的数据，且保留了所有维度（is_canceled 除外）。

4.1 随机森林

随机森林（Random Forest）是一种用于分类，回归和其他任务的集成学习方法，通过在训练时间内构建多个决策树并输出作为类的标签（分类）或个体树预测的平均值（回归）。随机森林中有许多的分类树。要将一个输入样本进行分类，需要将输入样本输入到每棵树中进行分类。每棵决策树都是一个分类器，那么对于一个输入样本，N 棵树会有 N 个分类结

果。而随机森林集成了所有的分类投票结果，将投票次数最多的类别指定为最终的输出。

随机森林中常用的参数有：特征选择方法（criterion）、决策树个数（n_estimators）和树的深度（max_depth）。接下来，对随机森林模型进行调参建模，结果如表 2 所示（红色数字代表最优结果）。

表 2 对随机森林进行调参建模的结果表

	criterion	n_estimators	max_depth	accuracy	f1-score
1	gini	1	5	0.734	0.533
2	gini	1	10	0.813	0.744
3	gini	1	20	0.868	0.818
4	gini	10	5	0.785	0.609
5	gini	10	10	0.858	0.777
6	gini	10	20	0.926	0.897
7	gini	50	5	0.778	0.582
8	gini	50	10	0.861	0.786
9	gini	50	20	0.931	0.903
10	gini	100	5	0.778	0.580
11	gini	100	10	0.865	0.794
12	gini	100	20	0.932	0.904
13	entropy	1	5	0.761	0.619
14	entropy	1	10	0.820	0.731
15	entropy	1	20	0.856	0.801
16	entropy	10	5	0.778	0.583
17	entropy	10	10	0.861	0.788
18	entropy	10	20	0.921	0.888
19	entropy	50	5	0.780	0.586
20	entropy	50	10	0.860	0.786
21	entropy	50	20	0.927	0.897
22	entropy	100	5	0.785	0.601
23	entropy	100	10	0.863	0.790
24	entropy	100	20	0.927	0.897

由表 2 可知，决策树第 12 组模型（criterion='gini', n_estimators=100, max_depth=20）的准确度与 f1-score 均达到最佳，分别为 0.932 和 0.904。

4.2 决策树

决策树（Decision Tree）是一种简单但广泛使用的分类器，它通过训练数据构建决策树，

对未知的数据进行分类。决策树的每个内部节点表示在一个属性上的测试，每个分枝代表该测试的一个输出，而每个树叶结点存放着一个类标号。在决策树算法中，ID3 基于信息增益作为属性选择的度量，C4.5 基于信息增益比作为属性选择的度量，CART 基于基尼指数作为属性选择的度量。

决策树中常用的参数有：特征选择方法（criterion）、特征划分点选择方法（splitter）和树的深度（max_depth）。接下来，对决策树模型进行调参建模，结果如表 3 所示（红色数字代表最优结果）。

表 3 对决策树进行调参建模的结果表

	criterion	splitter	max_depth	accuracy	f1-score
1	gini	best	5	0.791	0.662
2	gini	best	10	0.867	0.819
3	gini	best	20	0.929	0.905
4	gini	random	5	0.760	0.595
5	gini	random	10	0.823	0.735
6	gini	random	20	0.905	0.869
7	entropy	best	5	0.790	0.662
8	entropy	best	10	0.847	0.781
9	entropy	best	20	0.931	0.906
10	entropy	random	5	0.763	0.538
11	entropy	random	10	0.814	0.742
12	entropy	random	20	0.913	0.881

由表 3 可知，决策树第 9 组模型（criterion='entropy', splitter='best', max_depth=20）的准确度与 f1-score 均达到最佳，分别为 0.931 和 0.906。

4.3 AdaBoost

AdaBoost 英文"Adaptive Boosting"（自适应增强）的缩写，由 Yoav Freund 和 Robert Schapire 在 1995 年提出。它的自适应在于：前一个基本分类器分错的样本会得到加强，加权后的全体样本再次被用来训练下一个基本分类器。同时，在每一轮中加入一个新的弱分类器，直到达到某个预定的足够小的错误率或达到预先指定的最大迭代次数。

AdaBoost 中常用的参数是基分类器个数（n_estimators）。接下来，对 AdaBoost 模型进行调参建模，结果如表 4 所示（红色数字代表最优结果）。

表 4 对 AdaBoost 进行调参建模的结果表

	n_estimators	accuracy	f1-score
1	1	0.747	0.491
2	10	0.816	0.732
3	20	0.831	0.743

4	30	0.838	0.759
5	40	0.851	0.780
6	50	0.855	0.786
7	60	0.860	0.794
8	70	0.863	0.798
9	80	0.867	0.805
10	90	0.869	0.809
11	100	0.871	0.812

由表 4 可知，随着 n_estimators 参数的增大，模型的准确度与 f1-score 也随之增大，在 n_estimator 取到最大值 100 时，准确度与 f1-score 分别为 0.871 和 0.812。

4.4 逻辑回归

逻辑回归（Logistic Regression）是一种广义的线性回归分析模型，常用于数据挖掘，疾病自动诊断，经济预测等领域。它是基 Sigmoid 函数（又叫“S 型函数”）的有监督二类分类模型。

随机森林中常用的参数有：正则化系数 λ 的倒数 C、停止求解的标准 tol 等。接下来，对逻辑回归模型进行调参建模，结果如表 5 所示（红色数字代表最优结果）。

表 5 对逻辑回归进行调参建模的结果表

	C	tol	accuracy	f1-score
1	1	1e-3	0.954	0.935
2	1	1e-4	0.954	0.935
3	1	1e-5	0.954	0.936
4	10	1e-3	0.968	0.955
5	10	1e-4	0.968	0.956
6	10	1e-5	0.969	0.956
7	50	1e-3	0.972	0.961
8	50	1e-4	0.971	0.961
9	50	1e-5	0.972	0.961
10	100	1e-3	0.971	0.962
11	100	1e-4	0.971	0.962
12	100	1e-5	0.972	0.962

可以发现，逻辑回归的准确度与 f1-score 在 C 较小时与 C 的取值关系密切。而 C 等于 50 和 100 时，准确度与 f1-score 变化不大。此外，tol 参数对分类结果的影响很小。在 C=100 且 tol=1e-5 时，准确度与 f1-score 分别达到了 0.972 和 0.962。

4.5 人工神经网络

人工神经网络（Artificial Neural Network），是 20 世纪 80 年代以来人工智能领域兴起

的研究热点。它从信息处理角度对人脑神经元网络进行抽象， 建立某种简单模型，按不同的连接方式组成不同的网络。在工程与学术界也常直接简称为神经网络或类神经网络。神经网络是一种运算模型，由大量的节点（或称神经元）之间相互联接构成。每个节点代表一种特定的输出函数，称为激励函数（activation function）。每两个节点间的连接都代表一个对于通过该连接信号的加权值，称之为权重，这相当于人工神经网络的记忆。而网络自身通常都是对自然界某种算法或者函数的逼近，也可能是对一种逻辑策略的表达。

人工神经网络的训练结果受很多因素的影响，包括且不限于：模型种类、网络层数、batch size、激励函数、迭代次数（epochs）等。本实验中，我采用最简单的序贯模型，通过调整神经网络层数、batch size、epochs 等参数进行建模。

我设置了三组实验。第一组，将神经网络模型设置为 4 层，这三层依次设置 30、20、10、1 个神经元，其对应的激励函数分别为'relu'、'relu'、'relu'和'sigmoid'。第二组，将神经网络模型设置为 3 层，这三层依次设置 30、15、1 个神经元，其对应的激励函数分别为'relu'、'relu'和'sigmoid'。第三组，将神经网络模型设置为 3 层，这三层依次设置 30、15、1 个神经元，其对应的激励函数分别为'sigmoid'、'sigmoid'和'sigmoid'。对这三组模型，都设置 batch size 为[16, 32, 64, 128]，epochs 为[5, 10,20]，进行建模，并使用 adam 优化器进行编译，结果如表 6 所示（红色数字代表最优结果）。

表 6 对人工神经网络进行调参建模的结果表

Model	batch size	epochs	accuracy	f1-score
model = Sequential() model.add(Dense(units=30, activation='relu')) model.add(Dropout(0.5)) model.add(Dense(units=20, activation='relu')) model.add(Dropout(0.5)) model.add(Dense(units=10, activation='relu')) model.add(Dropout(0.5)) model.add(Dense(units=1, activation='sigmoid'))	16	5	0.949	0.928
	16	10	0.949	0.927
	16	20	0.959	0.943
	32	5	0.955	0.935
	32	10	0.941	0.916
	32	20	0.933	0.902
	64	5	0.930	0.897
	64	10	0.958	0.940
	64	20	0.963	0.949
	128	5	0.933	0.903
	128	10	0.958	0.941
	128	20	0.960	0.945
model = Sequential() model.add(Dense(units=30, activation='relu')) model.add(Dropout(0.5)) model.add(Dense(units=15,	16	5	0.947	0.925
	16	10	0.954	0.935
	16	20	0.959	0.941
	32	5	0.957	0.940
	32	10	0.959	0.942

activation='relu'))	32	20	0.961	0.945
model.add(Dropout(0.5))	64	5	0.944	0.920
model.add(Dense(units=1,	64	10	0.963	0.949
activation='sigmoid')	64	20	0.968	0.955
	128	5	0.923	0.885
	128	10	0.964	0.950
	128	20	0.972	0.961
model = Sequential()	16	5	0.870	0.794
model.add(Dense(units=30,	16	10	0.897	0.840
activation='sigmoid'))	16	20	0.919	0.879
model.add(Dropout(0.5))	32	5	0.857	0.770
model.add(Dense(units=15,	32	10	0.887	0.824
activation='sigmoid'))	32	20	0.908	0.860
model.add(Dropout(0.5))	64	5	0.848	0.753
model.add(Dense(units=1,	64	10	0.870	0.793
activation='sigmoid')	64	20	0.899	0.843
	128	5	0.844	0.754
	128	10	0.860	0.775
	128	20	0.889	0.826

可以看到，在第二组模型中，batch_size=64，epochs=20 时，人工神经网络分类效果最好，准确度与 f1-score 分别达到了 0.972 和 0.961。

4.6 模型评价

接下来，将对以上五种最优预测模型的准确度与 f1-score、ROC 曲线和 AUC 值进行比较。首先对准确度与 f1-score 进行比较，结果如图 13 所示。

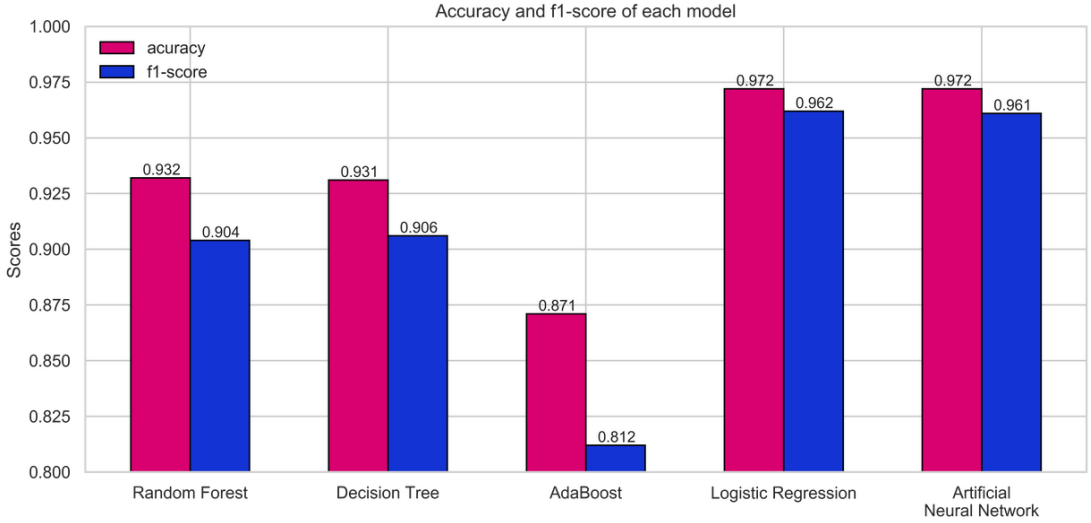

图 13 五种模型精确度与 f1-score 对比图

由图 13 可知，在上述五种模型中，逻辑回归模型的准确度与 f1-score 最好。人工神经网络模型准确度和逻辑回归一样，但 f1-score 排名第二。随机森林和决策树模型表现不相上下，准确度与 f1-score 均分别约为 0.93 和 0.9。五种模型中，AdaBoost 模型表现最糟糕。

然后，绘制 ROC 曲线并查看每种模型的 AUC 值，如图 14 所示。由图可知，这五种模型的 AUC 值都至少达到了 0.94，它们从高到低排名依次是逻辑回归（AUC=0.990）、决策树（AUC=0.979）、人工神经网络（AUC=0.960）、随机森林（AUC=0.953）和 AdaBoost（AUC=0.941）、

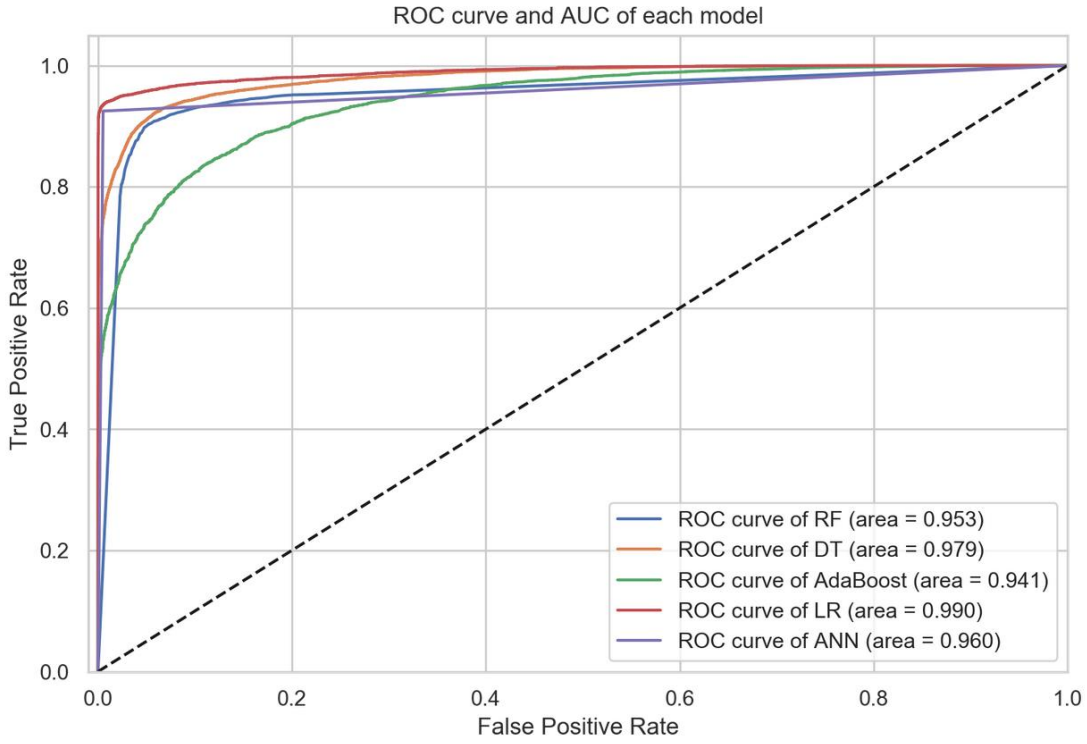

图 14 五种模型 AUC 值与 ROC 曲线图

由以上分析可知，这五种模型中，逻辑回归各项指标均为最优，因此判定逻辑回归（C=100，tol=1e-5）为最佳模型。

5 模型优化尝试

本节中，我采用两种方法尝试进行优化：一是对特征进行处理，清除无关字段；二是对上一节中表现较好的模型进行投票组合。

5.1 选取特定字段进行预测

特征选择(排序)对于数据挖掘来说非常重要。好的特征选择能够提升模型的性能，更能帮助我们理解数据的特点。这对进一步改善模型、算法都有着重要作用。

在 3.3 节图 10 中，我采用了 corr()方法，将所有特征对”is_canceled”标签的影响权重进行了排序与可视化。从排名第 11 位的特征”adr”开始，后面的所有特征对”is_canceled”标签的影响权重都小于 0.05。这些字段相对于排名靠前的特征来说，可以被认为是无关特征。为了对已有的预测模型进行优化，本节中，我仅保留了对”is_canceled”标签的影响权重大于 0.05

的特征，输入上节中的五个模型，并整理了最佳模型及配置。去除无关字段前后最优模型的数据对比如表 7 所示。

表 7 去除无关字段前后模型的对比表

模型种类	数据种类	最优参数配置	accuracy	f1-score
随机森林	全部字段	'criterion': 'gini', 'max_depth': 20, 'n_estimators': 100	0.932	0.904
	部分特定字段	'criterion': 'gini', 'max_depth': 20, 'n_estimators': 100	0.933	0.906
决策树	全部字段	'criterion': 'entropy', 'max_depth': 20, 'splitter': 'best'	0.931	0.906
	部分特定字段	'criterion': 'entropy', 'max_depth': 20, 'splitter': 'best'	0.930	0.906
AdaBoost	全部字段	n_estimators=100	0.871	0.812
	部分特定字段	n_estimators=100	0.871	0.812
逻辑回归	全部字段	C=100，tol=1e-5	0.972	0.962
	部分特定字段	C=50，tol=1e-5	0.972	0.961
人工神经网络	全部字段	表 6 的第二类模型，batch size=128，epochs=20	0.972	0.961
	部分特定字段	表 6 的第二类模型，batch size=64，epochs=20	0.970	0.958

由表 7 可知，保留部分与退订关联性强的字段与直接对全部字段进行建模预测相比，优化效果不明显。

5.2 使用投票法进行优化

投票法（voting）是集成学习里面针对分类问题的一种结合策略。基本思想是选择所有机器学习算法当中输出最多的那个类。分类的机器学习算法输出有两种类型：一种是直接输出类标签，另外一种是输出类概率，使用前者进行投票叫做硬投票(Majority/Hard voting)，使用后者进行分类叫做软投票(Soft voting)。sklearn 中的 VotingClassifier 是投票法的实现。

基于之前实验的结果，我挑选决策树、逻辑回归、随机森林这三个算法中最优参数配置的模型进行投票，分别采用硬投票和软投票的方式，调整模型权重分别为 1:1:1、1:2:1、3:4:3 进行实验。实验结果如表 8 所示（红色数字代表最优结果）。

表 8 用投票法进行优化模型的结果表

	投票方式	权重	accuracy	f1-score
1	hard	1:1:1	0.957	0.941
2	hard	1:2:1	0.959	0.941
3	hard	3:4:3	0.958	0.942

4	soft	1:1:1	0.965	0.953
5	soft	1:2:1	0.975	0.966
6	soft	3:4:3	0.972	0.961

可以看到，当采用软投票方式，将决策树、逻辑回归、随机森林权重比设置为 1:2:1 时，预测准确度和 f1-score 达到最佳，分别为 0.975 和 0.966，超过了之前最优的逻辑回归单一模型（精确度为 0.972，f1-score 为 0.962）。

接着，再查看投票器与三种基本分类模型的 ROC 曲线和 AUC 值对比，如图 15 所示。

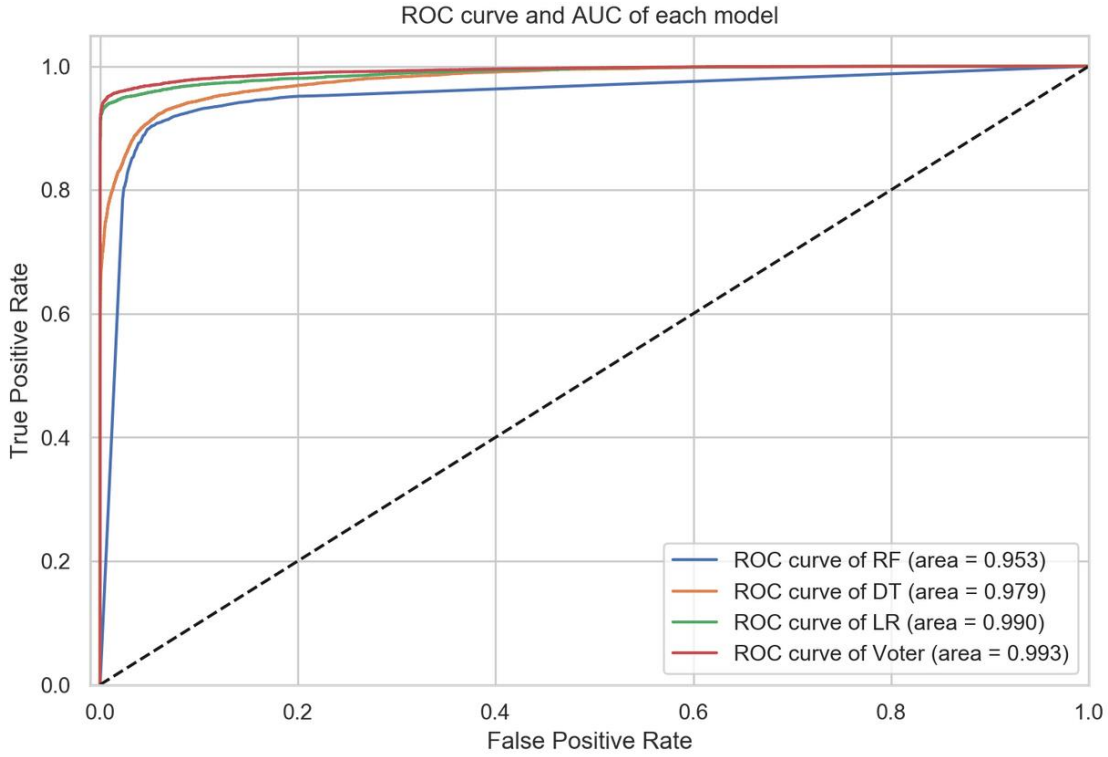

图 15 投票器与其基本分类模型的 AUC 值与 ROC 曲线图

由图 15 可知，投票器的 AUC 值比逻辑回归有了明显的提升，从 0.990 升为 0.993。从 ROC 曲线上也能看到，投票器的 ROC 曲线左上角比逻辑回归的 ROC 曲线略有突出。这说明投票器的预测结果要比三种分类器模型都要好。

6 总结

本次课设，我独立完成了一个酒店退订预测的数据挖掘项目。这个项目，连同之前历次上机实验，使我对数据挖掘有了一定的认识。

进行数据挖掘前，首先必须明确要解决的问题，接着要找到合适的数据进行研究，并根据数据集构建模型，最后不断优化模型，从而解决问题。这就是数据挖掘的基本流程。面对酒店订单数据集这个完全陌生的数据，我首先做的是将数据可视化，探索性地分析数据。然后，我做了一些预处理，如缺失值填充，标准化等。在对数据有了一定的理解后，我使用不同的算法去探索退订预测的方案，并使用 f1-score 等指标去评价模型。此外，我还通过调整参数配置，集成学习等方式优化模型，使模型不断逼近最优解，最后也取得了不错的预测效

果。这个解决问题的过程，让我体会到数据挖掘不仅仅是一门科学，也是一种非常有用的解决问题的思维方式。

除了数据挖掘的基本思路外，我还学习了一些常用的数据挖掘算法的原理与使用，例如分类，聚类，关联分析等。这些算法各有优势，需要针对不同问题灵活地进行调整，最终达到解决问题的目的。

也许未来我不会专门从事数据挖掘工作，但这门课带给我思维方式的进步将会使我终生受益。

参考文献：

[1] Antonio N, de Almeida A, Nunes L. Predicting hotel bookings cancellation with a machine learning classification model[C]. 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA). IEEE, 2017: 1049-1054.

[2] Antonio N, De Almeida A, Nunes L. Hotel booking demand datasets[J]. Data in brief, 2019, 22: 41-49.

[3] Hotel booking demand | Kaggle: https://www.kaggle.com/jessemostipak/hotel-booking-demand

附录：各模型实现代码及预测结果截图

随机森林模型代码与预测结果：

```
# Random Forest
tuned_parameters = [{'criterion': ['gini','entropy'], 'n_estimators': [1, 10, 50, 100],
                     'max_depth': [5, 10, 20]}]
scores = ['precision', 'recall']
classifier_RF = GridSearchCV(RandomForestClassifier(),
                    tuned_parameters, scoring='accuracy', cv=5) #交叉验证
classifier_RF.fit(X_train, y_train)
print('RF best parameters are:',classifier_RF.best_params_)
# y_score[1] = classifier_RF.fit(X_train, y_train).predict_proba(X_test)[:,1]
y_score[1] = classifier_RF.fit(X_train, y_train).predict(X_test)
print('RF_report:\n',classification_report(y_test, y_score[1]))
print("accuracy:", accuracy_score(y_test, y_score[1]))
print("f1-score:", f1_score(y_test, y_score[1], average='binary'))
```

```
RF best parameters are: {'criterion': 'gini', 'max_depth': 20, 'n_estimators': 100}
RF_report:
              precision    recall  f1-score   support

           0       0.92      0.97      0.95     18732
           1       0.95      0.86      0.90     11116

    accuracy                           0.93     29848
   macro avg       0.94      0.92      0.93     29848
weighted avg       0.93      0.93      0.93     29848

accuracy: 0.932122755293487
f1-score: 0.9044429770776341
```

决策树模型代码与预测结果：

```
# Decision Tree
tuned_parameters = [{'criterion': ['gini','entropy'], 'splitter': ['best','random'], 'max_depth': [5,10,20]}]
scores = ['precision', 'recall']
classifier_DT = GridSearchCV(DecisionTreeClassifier(),
                    tuned_parameters,scoring='accuracy', cv=5) #交叉验证
classifier_DT.fit(X_train, y_train)
print('DT best parameters are:',classifier_DT.best_params_)
# y_score[0] = classifier_DT.fit(X_train, y_train).predict_proba(X_test)[:,1]
y_score[0] = classifier_DT.fit(X_train, y_train).predict(X_test)
print('DT_report:\n',classification_report(y_test, y_score[0]))
print("accuracy:", accuracy_score(y_test, y_score[0]))
print("f1-score:", f1_score(y_test, y_score[0], average='binary'))
```

```
DT best parameters are: {'criterion': 'entropy', 'max_depth': 20, 'splitter': 'best'}
DT_report:
              precision    recall  f1-score   support

           0       0.94      0.95      0.95     18732
           1       0.92      0.90      0.91     11116

    accuracy                           0.93     29848
   macro avg       0.93      0.92      0.93     29848
weighted avg       0.93      0.93      0.93     29848

accuracy: 0.9308496381667113
f1-score: 0.9062329638379067
```

AdaBoost 模型代码与预测结果：

```
# AdaBoost
tuned_parameters = [{'n_estimators': [1,10,20,30,40,50,60,70,80,90,100]}]
scores = ['precision', 'recall']
classifier_AdaBoost = GridSearchCV(AdaBoostClassifier(),
                    tuned_parameters,scoring='accuracy', cv=5) #交叉验证
classifier_AdaBoost.fit(X_train, y_train)
print('AdaBoost best parameters are:',classifier_AdaBoost.best_params_)
# y_score[2] = classifier_AdaBoost.fit(X_train, y_train).decision_function(X_test)
y_score[2] = classifier_AdaBoost.fit(X_train, y_train).predict(X_test)
print('AdaBoost_report:\n',classification_report(y_test, y_score[2]))
print("accuracy:", accuracy_score(y_test, y_score[2]))
print("f1-score:", f1_score(y_test, y_score[2], average='binary'))
```

```
AdaBoost best parameters are: {'n_estimators': 100}
AdaBoost_report:
              precision    recall  f1-score   support

           0       0.86      0.95      0.90     18732
           1       0.89      0.75      0.81     11116

    accuracy                           0.87     29848
   macro avg       0.88      0.85      0.86     29848
weighted avg       0.87      0.87      0.87     29848

accuracy: 0.8712811578665237
f1-score: 0.812053615106154
```

逻辑回归模型代码与预测结果：

```
# Logistic Regression
tuned_parameters = [{'C': [1,10,50,100], 'tol': [1e-3,1e-4,1e-5]}]
scores = ['precision', 'recall']
classifier_LR = GridSearchCV(LogisticRegression(),
                    tuned_parameters,scoring='accuracy', cv=5) #交叉验证
classifier_LR.fit(X_train, y_train)
print('LR best parameters are:',classifier_LR.best_params_)
# y_score[3] = classifier_LR.fit(X_train, y_train).decision_function(X_test)
y_score[3] = classifier_LR.fit(X_train, y_train).predict(X_test)
print('LR_report:\n',classification_report(y_test, y_score[3]))
print("accuracy:", accuracy_score(y_test, y_score[3]))
print("f1-score:", f1_score(y_test, y_score[3], average='binary'))
```

```
LR best parameters are: {'C': 100, 'tol': 1e-05}
LR_report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98     18732
           1       0.99      0.93      0.96     11116

    accuracy                           0.97     29848
   macro avg       0.98      0.96      0.97     29848
weighted avg       0.97      0.97      0.97     29848

accuracy: 0.9723264540337712
f1-score: 0.9616349280074316
```

人工神经网络模型代码与预测结果：

```
model = Sequential()

#adding dropout layers for improved learning
model.add(Dense(units=30,activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(units=15,activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(units=1,activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])

#Putting early_stop in to prevent overfitting
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)

X_train = K.cast_to_floatx(X_train)
y_train = K.cast_to_floatx(y_train)
X_test = K.cast_to_floatx(X_test)
y_test = K.cast_to_floatx(y_test)
model.fit(x=X_train,
          y=y_train,
          batch_size=128,
          epochs=20,
          validation_data=(X_test, y_test), verbose=1,callbacks=[early_stop]
          )
y_score[4] = model.predict_classes(X_test)
print(classification_report(y_test,y_score[4]))
print("accuracy:", accuracy_score(y_test, y_score[4]))
print("f1-score:", f1_score(y_test, y_score[4], average='binary'))
```

```
Epoch 1/20
700/700 [==============================] - 1s 2ms/step - loss: 0.6149 - accuracy: 0.6823 - val_loss: 0.4295 - val_accuracy: 0.7947
Epoch 2/20
700/700 [==============================] - 1s 2ms/step - loss: 0.4605 - accuracy: 0.7903 - val_loss: 0.3839 - val_accuracy: 0.8239
Epoch 3/20
700/700 [==============================] - 1s 2ms/step - loss: 0.4088 - accuracy: 0.8199 - val_loss: 0.3357 - val_accuracy: 0.8554
Epoch 4/20
700/700 [==============================] - 1s 2ms/step - loss: 0.3573 - accuracy: 0.8520 - val_loss: 0.2692 - val_accuracy: 0.8970
Epoch 5/20
700/700 [==============================] - 1s 2ms/step - loss: 0.3003 - accuracy: 0.8829 - val_loss: 0.2232 - val_accuracy: 0.9154

Epoch 6/20
700/700 [==============================] - 1s 2ms/step - loss: 0.2585 - accuracy: 0.9023 - val_loss: 0.1825 - val_accuracy: 0.9411
Epoch 7/20
700/700 [==============================] - 1s 2ms/step - loss: 0.2284 - accuracy: 0.9169 - val_loss: 0.1604 - val_accuracy: 0.9486
Epoch 8/20
700/700 [==============================] - 1s 2ms/step - loss: 0.2031 - accuracy: 0.9277 - val_loss: 0.1420 - val_accuracy: 0.9556
Epoch 9/20
700/700 [==============================] - 1s 2ms/step - loss: 0.1853 - accuracy: 0.9352 - val_loss: 0.1305 - val_accuracy: 0.9578
Epoch 10/20
700/700 [==============================] - 1s 2ms/step - loss: 0.1738 - accuracy: 0.9392 - val_loss: 0.1221 - val_accuracy: 0.9667
Epoch 11/20
700/700 [==============================] - 1s 2ms/step - loss: 0.1666 - accuracy: 0.9424 - val_loss: 0.1168 - val_accuracy: 0.9649
Epoch 12/20
700/700 [==============================] - 1s 2ms/step - loss: 0.1617 - accuracy: 0.9434 - val_loss: 0.1225 - val_accuracy: 0.9599
Epoch 13/20
700/700 [==============================] - 1s 2ms/step - loss: 0.1567 - accuracy: 0.9455 - val_loss: 0.1143 - val_accuracy: 0.9638
Epoch 14/20
700/700 [==============================] - 1s 2ms/step - loss: 0.1537 - accuracy: 0.9475 - val_loss: 0.1085 - val_accuracy: 0.9665
Epoch 15/20
700/700 [==============================] - 1s 2ms/step - loss: 0.1528 - accuracy: 0.9473 - val_loss: 0.1107 - val_accuracy: 0.9647
Epoch 16/20
700/700 [==============================] - 1s 2ms/step - loss: 0.1495 - accuracy: 0.9491 - val_loss: 0.1065 - val_accuracy: 0.9660
Epoch 17/20
700/700 [==============================] - 1s 2ms/step - loss: 0.1482 - accuracy: 0.9499 - val_loss: 0.1126 - val_accuracy: 0.9636
Epoch 18/20
700/700 [==============================] - 1s 2ms/step - loss: 0.1451 - accuracy: 0.9508 - val_loss: 0.1000 - val_accuracy: 0.9697
Epoch 19/20
700/700 [==============================] - 1s 2ms/step - loss: 0.1413 - accuracy: 0.9522 - val_loss: 0.1018 - val_accuracy: 0.9680
Epoch 20/20
700/700 [==============================] - 1s 2ms/step - loss: 0.1432 - accuracy: 0.9517 - val_loss: 0.0958 - val_accuracy: 0.9720
              precision    recall  f1-score   support

         0.0       0.96      1.00      0.98     18732
         1.0       1.00      0.93      0.96     11116

    accuracy                           0.97     29848
   macro avg       0.98      0.96      0.97     29848
weighted avg       0.97      0.97      0.97     29848

accuracy: 0.972024926293219
f1-score: 0.9611139570623575
```

决策树、逻辑回归、随机森林投票器模型代码与预测结果：

```
# 投票，改善分类器性能
classifier_voting = VotingClassifier(estimators=[('DT', classifier_DT),
                ('LR', classifier_LR), ('RF', classifier_RF)], voting='soft', weights=[1,2,1])
classifier_voting = classifier_voting.fit(X_train, y_train)
voting_score = classifier_voting.fit(X_train, y_train).predict(X_test)
print('voting_report----soft:\n',classification_report(y_test, voting_score))
print("accuracy:", accuracy_score(y_test, voting_score))
print("f1-score:", f1_score(y_test, voting_score, average='binary'))
```

```
voting_report----soft:
              precision    recall  f1-score   support

         0.0       0.96      1.00      0.98     18732
         1.0       1.00      0.94      0.97     11116

    accuracy                           0.98     29848
   macro avg       0.98      0.97      0.97     29848
weighted avg       0.98      0.98      0.98     29848

accuracy: 0.975375234521576
f1-score: 0.9659422640285436
```