

《机器学习A》

课程报告

专业班级：195221

学生姓名：钱晓楠

学生学号：20221003675

指导老师：刘超

**中国地质大学计算机学院**

**2024年 12月**

目 录

[基本任务1.1 5](#_Toc1677)

[一、 问题理解与分析 5](#_Toc29173)

[二、 算法原理阐述 5](#_Toc18258)

[2.1贝叶斯定理 5](#_Toc11947)

[2.2朴素贝叶斯分类器 6](#_Toc30660)

[2.3 拉普拉斯修正 6](#_Toc16062)

[2.4算法步骤 7](#_Toc25588)

[三、 算法设计思路 8](#_Toc8976)

[四、 数据预处理 8](#_Toc16879)

[五、 具体实现 8](#_Toc22380)

[5.1函数介绍 8](#_Toc27797)

[5.2 代码整体结构 9](#_Toc26727)

[5.2 代码重要部分 10](#_Toc20094)

[5.3 实验流程 11](#_Toc11591)

[六、 实验结果分析 12](#_Toc17588)

[七、 解决问题及主要收获 14](#_Toc28226)

[7.1 遇到的问题 14](#_Toc16521)

[7.2 主要收获 15](#_Toc16785)

[八、 参考来源 15](#_Toc26836)

[基本任务1.2 15](#_Toc32292)

[一、 问题理解与分析 15](#_Toc15961)

[二、 算法原理阐述 16](#_Toc22785)

[2.1 KNN算法原理 16](#_Toc9253)

[2.2 决策树算法原理 16](#_Toc10899)

[2.2.1 树结构概述 17](#_Toc29738)

[2.2.2 决策树构建过程 17](#_Toc883)

[2.2 基尼指数算法原理阐述 18](#_Toc18784)

[三、 算法设计思路 18](#_Toc13422)

[四、 数据预处理 18](#_Toc13359)

[五、 具体实现 19](#_Toc19421)

[5.1 代码结构 19](#_Toc29427)

[5.2 实验流程 20](#_Toc23169)

[六、 实验结果分析 21](#_Toc22487)

[6.1 结果可视化 21](#_Toc9349)

[6.2 KNN模型 22](#_Toc3678)

[6.3 决策树模型 24](#_Toc18263)

[6.4 KNN模型与决策树模型的比较 25](#_Toc17999)

[七、 解决问题主要收获 25](#_Toc25457)

[7.1 解决的问题 25](#_Toc14045)

[7.2 主要收获 25](#_Toc4431)

[八、 参考来源 26](#_Toc17480)

[综合任务二 26](#_Toc25375)

[一、问题理解与分析 26](#_Toc9031)

[1.1 问题背景 26](#_Toc242)

[1.1.1 信用卡欺诈检测的背景和重要性 26](#_Toc28922)

[1.1.2 数据集介绍 26](#_Toc7031)

[1.2 目标与挑战 28](#_Toc11062)

[二、 算法原理阐述 28](#_Toc4484)

[2.1 神经网络（Neural Network） 28](#_Toc5224)

[2.2 XGBoost 30](#_Toc18174)

[2.2.1 XGBoost 预测原理 31](#_Toc10088)

[2.2.2 XGBoost 的目标函数 31](#_Toc24537)

[2.2.3 XGBoost的优化过程 31](#_Toc8934)

[2.2.4 树的分裂增益 32](#_Toc26700)

[2.2.5 叶子节点的权重计算 32](#_Toc12645)

[2.3 LightGBM 32](#_Toc31595)

[2.3.1 LightGBM 预测原理 33](#_Toc6617)

[2.3.2 LightGBM 的目标函数 33](#_Toc17344)

[2.3.3 核心优化点 33](#_Toc1634)

[2.3.4 叶子节点权重计算 34](#_Toc19881)

[2.3.5 LightGBM的正则化与控制参数 34](#_Toc6591)

[2.4 AutoEncoder 34](#_Toc8562)

[2.4.1 AutoEncoder 的基本结构 34](#_Toc4454)

[2.4.2 AutoEncoder 的目标函数 35](#_Toc2961)

[2.4.3 编码器与解码器的网络结构 35](#_Toc24943)

[三、算法设计思路 36](#_Toc27497)

[3.1总体思路 36](#_Toc24747)

[3.2模型组合与选择 36](#_Toc25984)

[3.2.1 课内模型 36](#_Toc3745)

[3.2.2 课外模型 36](#_Toc31759)

[3.3评价指标 37](#_Toc2631)

[3.3.1 ROC曲线与AUC值 37](#_Toc26953)

[3.3.2 分类报告（Classification Report） 37](#_Toc18352)

[3.3.3 混淆矩阵（Confusion Matrix） 37](#_Toc12513)

[四、 数据预处理 38](#_Toc17419)

[4.1初步检查与处理 38](#_Toc19077)

[4.2特征相关性分析 39](#_Toc12959)

[4.3异常值处理 41](#_Toc5483)

[4.3.1 IQR方法的基本原理 41](#_Toc12972)

[4.3.2 异常值处理与分析 41](#_Toc1323)

[4.4数据采样 42](#_Toc23899)

[4.2.1 随机下采样 42](#_Toc14964)

[4.2.2 SMOTE过采样 43](#_Toc13344)

[4.5降维 44](#_Toc3681)

[4.5.1 PCA（主成分分析） 44](#_Toc31551)

[4.5.2 t-SNE（t分布随机邻域嵌入） 44](#_Toc3446)

[五、 具体实现 45](#_Toc10571)

[5.1 实验总体设计与流程 45](#_Toc1937)

[5.2 课内模型 46](#_Toc7133)

[5.2.1 KNN（K-Nearest Neighbor）： 46](#_Toc17489)

[5.2.2 基于拉普拉斯变换的贝叶斯分类器 47](#_Toc731)

[5.2.3 随机森林（Random Forest） 48](#_Toc10174)

[5.3 课外模型 49](#_Toc16124)

[5.3.1 神经网络（Neural Network） 49](#_Toc12607)

[5.3.2 XGBoost（eXtreme Gradient Boosting） 51](#_Toc5489)

[5.3.3 LightGBM（Light Gradient Boosting Machine） 52](#_Toc20249)

[5.3.4 AutoEncoder + 逻辑回归 54](#_Toc27101)

[六、实验结果分析 56](#_Toc4875)

[6.1模型评估 56](#_Toc30977)

[6.1.1 不采样数据 56](#_Toc30927)

[6.1.2 随机下采样数据 57](#_Toc8643)

[6.1.3 SMOTE过采样数据 58](#_Toc13694)

[6.2采样方法影响 60](#_Toc30587)

[七、解决问题及主要收获 60](#_Toc32550)

[7.1问题解决 60](#_Toc28593)

[7.2主要收获 61](#_Toc30235)

[八、参考来源 61](#_Toc20139)

# 基本任务1.1

# 问题理解与分析

实现拉普拉斯修正的朴素贝叶斯分类器，以西瓜数据集 3.0 为训练集，对

“测 1”进行判别，“测1”如下：

表 1 西瓜数据集3.0“测1”

| 编号 | 色泽 | 根蒂 | 敲声 | 纹理 | 脐部 | 触感 | 密度 | 含糖率 | 好瓜 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 测1 | 青绿 | 蜷缩 | 浊响 | 清晰 | 凹陷 | 硬滑 | 0.697 | 0.460 | ？ |

朴素贝叶斯分类器是一种基于贝叶斯定理的简单概率分类器，它假设特征之间相互独立。拉普拉斯修正用于处理训练集中某些特征值未出现的情况，避免概率为零的问题。在解决该问题时，首先要根据拉普拉斯修正的朴素贝叶斯分类器的算法原理编程实现基本模型，再对西瓜数据集 3.0进行数据处理，并用处理后的数据进行模型训练，然后利用训练好的模型（已经实现参数更新）对“测1”数据进行判别，模型能够输出“测1”分别为好瓜与坏瓜的概率，从而可以判别出“测1”的类别。

# 算法原理阐述

拉普拉斯修正的贝叶斯分类器（Laplace-corrected Naive Bayes Classifier）是一种用于分类任务的统计模型，它基于贝叶斯定理和特征条件独立假设。拉普拉斯修正主要用于解决朴素贝叶斯分类器中可能出现的零概率问题，从而提高模型的鲁棒性和准确性。

## 2.1贝叶斯定理

贝叶斯定理描述了在已知某些条件下，事件发生的概率。对于分类问题，贝叶斯定理可以表示为：



其中：

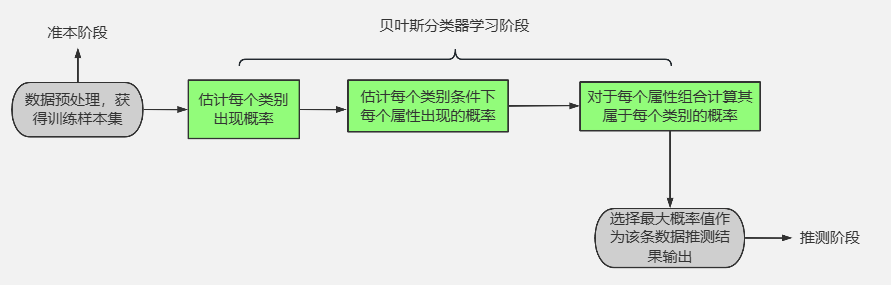
是在给定特征的条件下，类别的后验概率。

是类别下特征 的条件概率（似然）。

是类别的先验概率。

是特征的边缘概率，通常在分类问题中作为归一化常数。

## 2.2朴素贝叶斯分类器



朴素贝叶斯分类器假设所有特征在给定类别下是条件独立的，即：



其中是特征向量的第个特征。

## 2.3 拉普拉斯修正

在实际应用中，某些特征值可能在训练数据中没有出现过，导致条件概率。这会导致整个类别的后验概率为零，从而影响分类结果。拉普拉斯修正通过在分子和分母上分别加上一个常数来避免这种情况。

对于离散特征，拉普拉斯修正后的条件概率计算公式为：



其中：

是特征取值为且类别为的样本数。

是类别为 的样本总数。

是特征 可能取值的总数。

对于连续特征，通常假设其服从正态分布，拉普拉斯修正主要体现在样本量较小的情况下，通过增加样本量来提高估计的稳定性。

## 2.4算法步骤

以下是基于拉普拉斯修正的贝叶斯模型算法步骤：

#### 1.计算先验概率



其中是类别总数。

#### 计算条件概率

**离散特征：**



**连续特征：**

假设特征服从正态分布，计算均值和标准差：





其中是指示函数，表示样本的类别是否为。

#### 计算后验概率



对于连续特征，使用正态分布的概率密度函数：



#### 分类

选择后验概率最大的类别作为预测结果：



# 算法设计思路

首先，设计一个 `getDataSet` 函数从CSV文件中读取西瓜数据集，并提取特征和标签信息。

接着，设计一个 `naiveBayesClassifier` 函数构建朴素贝叶斯分类器，其核心任务是计算每个特征的条件概率以及类别的先验概率，计算了“好瓜”和“坏瓜”的先验概率，并将所有概率信息存储在一个嵌套字典中，供后续预测使用。

对于离散型特征，如“色泽”和“根蒂”，要计算它的条件概率，需要设计一个 `cntProLap` 函数，计算其在“好瓜”和“坏瓜”类别下的条件概率，并通过拉普拉斯平滑避免零概率问题；对于连续型特征，如“密度”和“含糖率”，假设其服从正态分布，计算其均值和方差以描述其分布特性。

最后进行预测，设计一个 `predict` 函数利用构建好的概率字典，计算给定样本在“好瓜”和“坏瓜”类别下的联合概率，并选择概率较大的类别作为预测结果。对于连续型特征，设计一个 `NormDist` 函数根据正态分布公式计算其概率密度。

评估分类器的准确率时，通过遍历数据集并比较预测结果与真实标签，计算分类正确的样本比例。

# 数据预处理

本次实验所采用的数据集为watermelon3\_0\_Ch.csv，文件包含了西瓜的多个特征及其对应的类别标签。数据预处理分为“数据读取”、“特征提取”、“新建特征字典”与“数据分割”等步骤。

首先进行数据读取，使用 pandas 库的 read\_csv 函数读取CSV文件，并将数据存储在 DataFrame 中。通过 df.values[1:, 1:] 提取样本数据，保留特征和标签数据。

定义特征列表 features来进行特征提取，存储西瓜的多个特征，如“色泽”、“根蒂”、“敲声”、“纹理”、“脐部”、“触感”、“密度”和“含糖率”。

为了方便处理离散型特征，定义一个特征字典 featureDic，其中每个特征对应其可能的取值。例如，“色泽”特征可能的取值为“浅白”、“青绿”和“乌黑”。

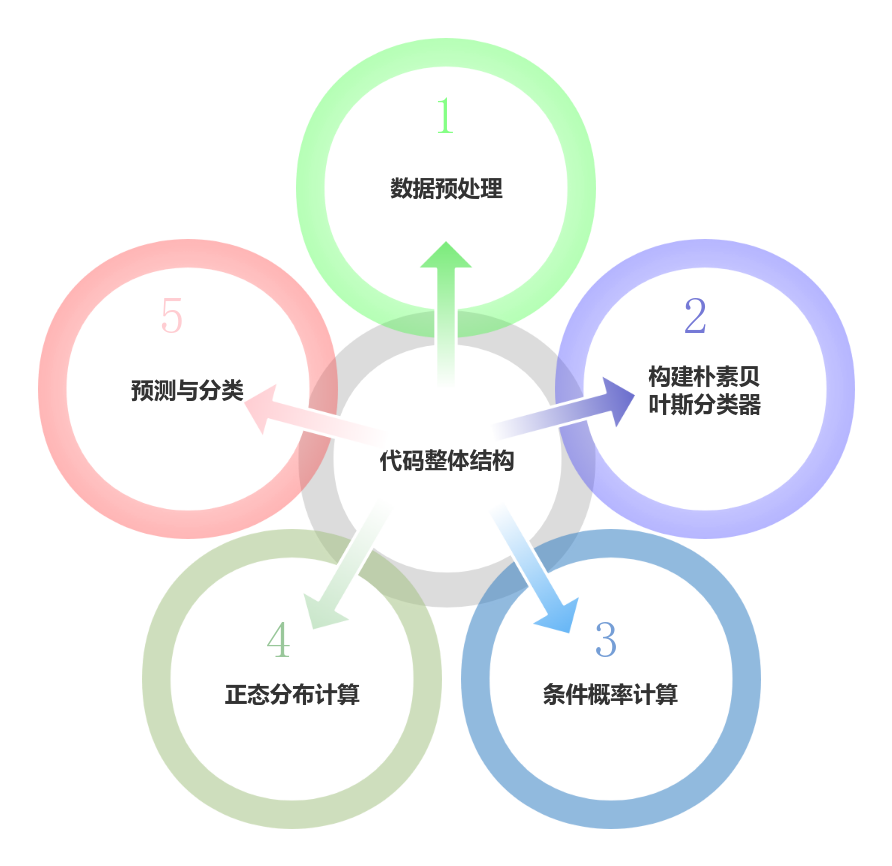
最后将数据集分成特征数据和标签数据，用于模型训练。

# 具体实现

## 5.1函数介绍

| **函数名** | **输入参数** | **返回值** | **作用/功能** |
| --- | --- | --- | --- |
| getDataSet | 无 | dataSet, features | 读取CSV文件，返回样本数据和特征列表。 |
| cntProLap | dataSet, index, value, classLabel, N | 条件概率 | 计算给定特征值和类别的条件概率，使用拉普拉斯修正。 |
| naiveBayesClassifier | dataSet, features | dict | 训练朴素贝叶斯分类器，计算并返回每个特征的条件概率和先验概率。 |
| NormDist | mean, var, xi | 正态分布概率密度值 | 计算给定均值、方差和特征值的正态分布概率密度值。 |
| predict | data, features, bayesDis | pGood, pBad, retClass | 使用训练好的朴素贝叶斯分类器对单个样本进行预测，返回好瓜和坏瓜的概率及预测类别。 |
| calcAccRate | dataSet, features, bayesDis | 准确率 | 计算模型在测试集上的准确率。 |

## 5.2 代码整体结构



“数据预处理”模块负责读取数据、处理数据，为模型训练做好准备。

“构建朴素贝叶斯”模块根据训练数据集，计算每个特征的条件概率以及类别的先验概率，最终将这些信息存储在一个字典中，供后续预测使用。模块首先判断特征是离散型还是连续型特征，然后通过拉普拉斯平滑和正态分布假设确保概率计算的鲁棒性，最终为后续的预测提供概率基础。

“条件概率计算”模块通过 cntProLap 函数计算离散型特征在给定类别下的条件概率，并使用拉普拉斯平滑处理，避免概率为零的情况。

“正态分布计算”模块通过 NormDist 函数计算连续型特征在给定类别下的概率密度，假设连续型特征服从正态分布。

“预测与分类”模块通过 predict 函数对给定样本进行分类，计算其为“好瓜”和“坏瓜”的概率，并选择概率较大的类别作为预测结果。

## 5.2 代码重要部分

naiveBayesClassifier 函数的关键代码及其详细解释：

#### 1.初始化字典

创建一个空字典 dict，用于存储所有特征的条件概率和类别的先验概率。

|  |
| --- |
| dict = {} |

#### 2.遍历特征

遍历特征列表 features，获取当前特征在列表中的索引 index。在字典 dict 中为当前特征创建一个空字典，用于存储该特征的条件概率。

|  |
| --- |
| for feature in features:  index = features.index(feature)  dict[feature] = {} |

#### 3.处理离散型特征

如果特征不是“密度”或“含糖率”，则认为是离散型特征。从 featureDic 中获取该特征的所有可能取值 featIList。对于每个取值，调用 cntProLap 函数计算其在“好瓜”和“坏瓜”类别下的条件概率，并使用拉普拉斯平滑处理。将计算结果存储到字典中，格式为：

dict[feature][value] = {"是": PisCond, "否": pNoCond}

|  |
| --- |
| if feature != '密度' and feature != '含糖率': # 为离散值  featIList = featureDic[feature] # 该特征的所有特征值存在featIList中  for value in featIList:  PisCond = cntProLap(dataSet, index, value, '是', len(featIList))  pNoCond = cntProLap(dataSet, index, value, '否', len(featIList))  dict[feature][value] = {}  dict[feature][value]["是"] = PisCond  dict[feature][value]["否"] = pNoCond |

#### 4.处理连续型特征

如果特征是“密度”或“含糖率”，则认为是连续型特征。对于每个类别（“是”或“否”），提取该类别下该特征的所有值，并将其转换为浮点数数组。计算该特征值的均值和方差，用于描述其正态分布。将均值和方差存储到字典中，格式为：

dict[feature][labelStr] = {"平均值": aver, "方差": var}

|  |
| --- |
| else: # 为连续值  for label in ['是', '否']:  dataExtra = dataSet[dataSet[:, -1] == label]  extr = dataExtra[:, index].astype("float64") # 取出该特征的所有值（连续值，数字等），作为数组  aver = extr.mean() # 求平均  var = extr.var() # 求方差  labelStr = ""  if label == '是':  labelStr = '是'  else:  labelStr = '否'  dict[feature][labelStr] = {}  dict[feature][labelStr]["平均值"] = aver  dict[feature][labelStr]["方差"] = var |

#### 5.计算先验概率

统计训练数据集中“好瓜”和“坏瓜”的数量。使用拉普拉斯平滑计算类别的先验概率，公式为：

P(类别) = (类别数量 + 1) / (总样本数 + 类别数)

将先验概率存储到字典中，格式为：

dict["好瓜"] = {"是": P(是), "否": P(否)}

|  |
| --- |
| length = len(dataSet) # 样本数量  classLabels = dataSet[:, -1].tolist()dict["好瓜"] = {}dict["好瓜"]['是'] = (classLabels.count('1') + 1) / (float(length) + 2) # 计算先验概率，这里的类别数为2，所以分母加2dict["好瓜"]['否'] = (classLabels.count('0') + 1) / (float(length) + 2) |

## 5.3 实验流程

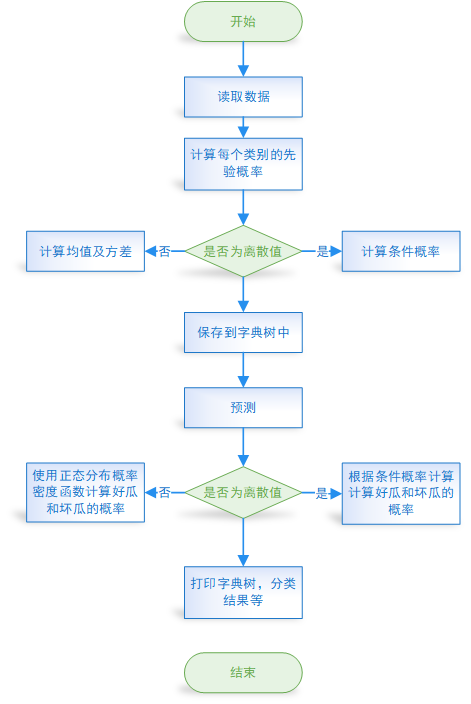
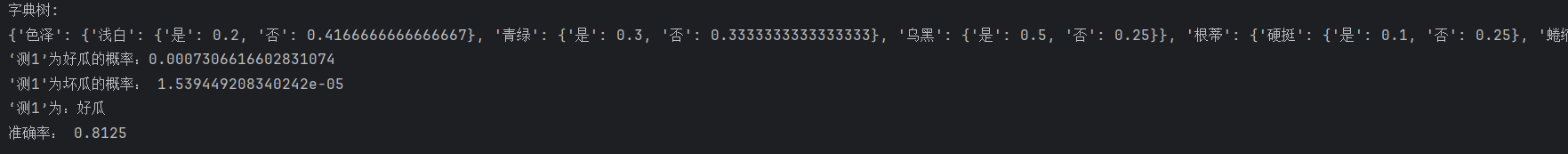


图 1 实验流程图

# 实验结果分析

实验结果如下：



字典树:

{'色泽': {'浅白': {'是': 0.2, '否': 0.4166666666666667}, '青绿': {'是': 0.3, '否': 0.3333333333333333}, '乌黑': {'是': 0.5, '否': 0.25}}, '根蒂': {'硬挺': {'是': 0.1, '否': 0.25}, '蜷缩': {'是': 0.5, '否': 0.3333333333333333}, '稍蜷': {'是': 0.4, '否': 0.4166666666666667}}, '敲声': {'沉闷': {'是': 0.3, '否': 0.3333333333333333}, '浊响': {'是': 0.6, '否': 0.4166666666666667}, '清脆': {'是': 0.1, '否': 0.25}}, '纹理': {'清晰': {'是': 0.7, '否': 0.25}, '模糊': {'是': 0.1, '否': 0.3333333333333333}, '稍糊': {'是': 0.2, '否': 0.4166666666666667}}, '脐部': {'凹陷': {'是': 0.5, '否': 0.25}, '平坦': {'是': 0.1, '否': 0.4166666666666667}, '稍凹': {'是': 0.4, '否': 0.3333333333333333}}, '触感': {'硬滑': {'是': 0.6666666666666666, '否': 0.6363636363636364}, '软粘': {'是': 0.3333333333333333, '否': 0.36363636363636365}}, '密度': {'是': {'平均值': 0.5561428571428572, '方差': 0.01421526530612245}, '否': {'平均值': 0.49611111111111117, '方差': 0.03370254320987655}}, '含糖率': {'是': {'平均值': 0.2528571428571429, '方差': 0.004822122448979593}, '否': {'平均值': 0.1542222222222222, '方差': 0.010328617283950618}}, '好瓜': {'是': 0.05555555555555555, '否': 0.05555555555555555}}

|  |  |
| --- | --- |
| ‘测1’为好瓜的概率 | 0.0007306616602831074 |
| ‘测1’为坏瓜的概率 | 1.539449208340242e-05 |
| ‘测1’类别 | 好瓜 |
| 准确率 | 0.8125 |

表 2 ‘测1’判别结果

由上表结果可知，‘测1’为好瓜与坏瓜的概率都偏小，但好瓜的概率更大，所以‘测1’更偏向于好瓜。至于为什么概率值那么小，也许是因为贝叶斯模型计算的是联合概率，且特征数量较多，导致最终概率值较小。

分类器在整个数据集上的准确率为 81.25%，表明模型的分类性能较好。

但仍然有一定的改进空间，由于我们的西瓜数据集数据量小，后续可以尝试增加训练数据量，或进一步优化特征工程（如特征选择或特征变换），来进一步提高模型的准确率。对于连续型特征，可以验证其是否真正服从正态分布，必要时采用其他分布假设。

对存储了条件概率的字典进行可视化得到如下两个表格，分别为离散特征的直方图和连续特征的概率分布图。

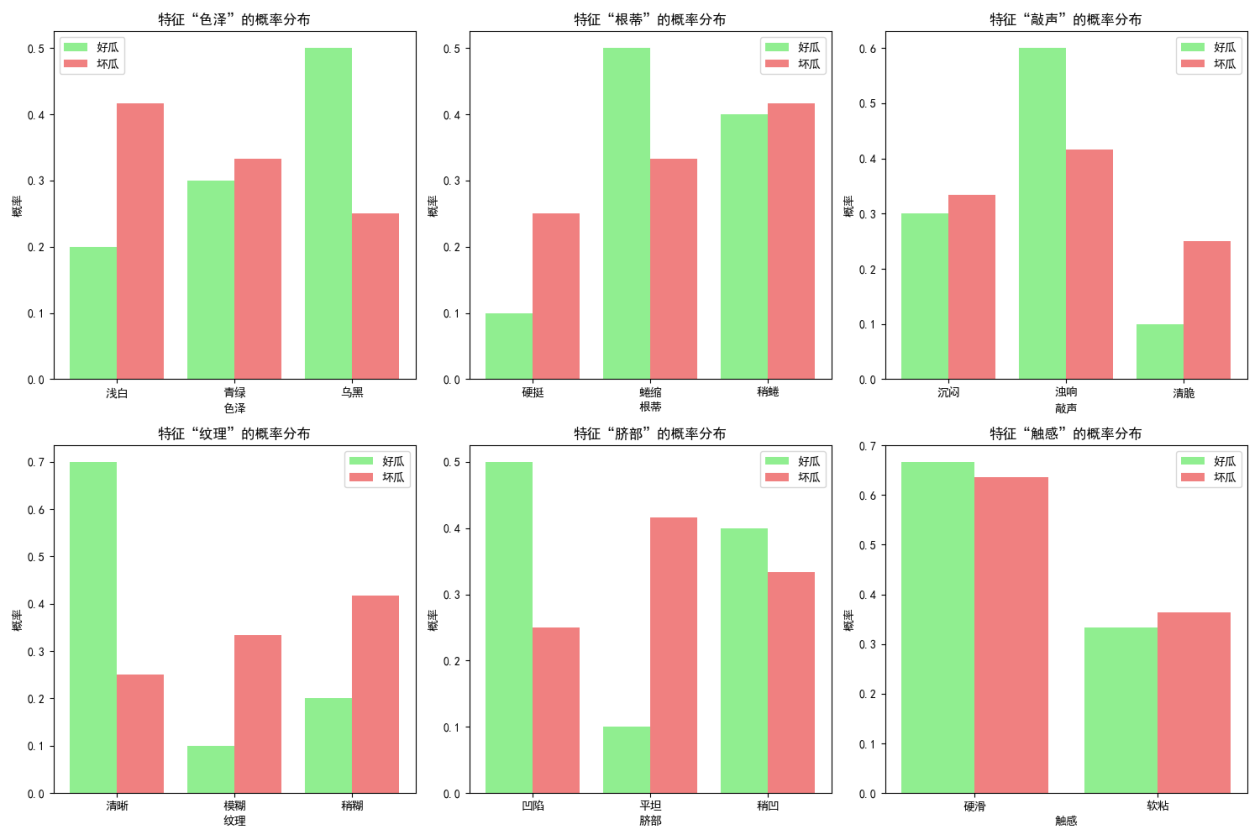


图 2 离散特征条件概率直方图

离散特征的条件概率清晰地反映了特征值与类别之间的关系，为分类器提供了重要的区分依据。离散特征包括“色泽”、“根蒂”、“敲声”、“纹理”、“脐部”和“触感”，每个特征有若干取值。字典树中存储了每个特征值在“好瓜”和“坏瓜”类别下的条件概率。

由离散特征条件概率直方图可以看到，条件概率清晰地反映了特征值与类别之间的关系。例如，“纹理=清晰”在“好瓜”类别下的概率（0.7）显著高于“坏瓜”类别下的概率（0.25），表明“清晰”是“好瓜”的重要特征；而“浅白”在“坏瓜”类别下的概率（0.4167）高于“好瓜”类别下的概率（0.2），表明“浅白”更可能与“坏瓜”相关。这些概率值通过拉普拉斯平滑处理，避免了零概率问题，为分类器提供了明确的区分依据，使得离散特征在分类决策中起到了关键作用。

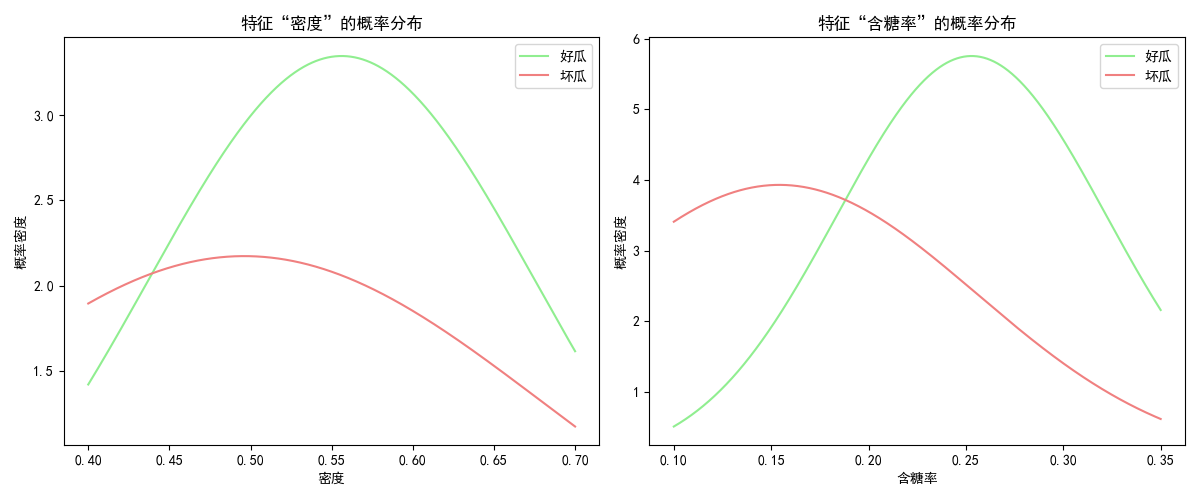


图 3 连续特征条件概率直方图

连续特征的均值和方差用于计算正态分布的概率密度，为分类器提供了连续型特征的区分依据。

由连续特征条件概率直方图可以看到，其均值和方差描述了数据在“好瓜”和“坏瓜”类别下的分布特性。例如，“好瓜”的密度均值（0.5561）高于“坏瓜”的密度均值（0.4961），且方差较小，表明“好瓜”通常密度较大且分布集中；同样，“好瓜”的含糖率均值（0.2529）高于“坏瓜”的含糖率均值（0.1542），且方差较小，表明“好瓜”通常含糖率较高且分布稳定。通过正态分布假设，分类器能够计算连续特征的概率密度，从而在预测时综合考虑这些特征的影响，增强了模型的分类能力。离散特征和连续特征的结合使得分类器能够充分利用数据信息，达到较高的准确率。

# 解决问题及主要收获

## 7.1 遇到的问题

在实验过程中，我遇到了两个主要问题。

在最开始的数据读取方面，我的数据读入格式有误。最初在读取CSV文件时，由于对 pandas 库的 read\_csv 函数使用不熟悉，导致数据读入后格式不符合预期，尤其是表头和索引的处理出现了混乱。经过查阅文档和调试，我学会了如何正确指定参数（如 header 和 index\_col），最终成功提取了所需的特征和标签数据。

在实现朴素贝叶斯分类器时，我对连续型特征的处理方式理解不够深入，最初错误地将其视为离散型特征进行处理，导致预测结果不准确。通过重新学习概率论中的正态分布知识，并参考相关代码示例，我修正了这一问题，采用均值和方差来描述连续型特征的分布，最终使分类器的预测性能得到了显著提升。

## 7.2 主要收获

通过这次实验，我收获颇丰，不仅加深了对朴素贝叶斯算法的理解，也提升了自己的编程能力和问题解决能力。首先，我学会了如何从实际数据出发，构建一个完整的机器学习模型，包括数据读取、特征处理、模型训练和性能评估。其次，我对概率论中的条件概率和正态分布有了更直观的认识，尤其是在连续型特征的处理上，理解了如何通过均值和方差来描述数据的分布特性。此外，在调试代码的过程中，我深刻体会到细节的重要性，例如数据格式的规范和算法的正确实现，这些细节往往决定了模型的最终性能。

最让我感到欣慰的是，通过不断尝试和修正错误，我最终成功实现了分类器，并达到了较高的准确率。这种从问题到解决的过程，不仅让我收获了知识，也增强了我的自信心。我深刻认识到，机器学习不仅仅是理论的学习，更需要动手实践和不断调试。这次实验让我对机器学习产生了更浓厚的兴趣，也让我更加坚定了继续学习和探索的决心。

# 参考来源

1. [【朴素贝叶斯】深入浅出讲解朴素贝叶斯算法（公式、原理）-CSDN博客](https://blog.csdn.net/kevinjin2011/article/details/125099177?ops_request_misc=%7B%22request%5Fid%22%3A%22dd2cc224d89ec16873caffbb3d05e596%22%2C%22scm%22%3A%2220140713.130102334..%22%7D&request_id=dd2cc224d89ec16873caffbb3d05e596&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-1-125099177-null-null.142^v100^pc_search_result_base4&utm_term=%E6%9C%B4%E7%B4%A0%E8%B4%9D%E5%8F%B6%E6%96%AF&spm=1018.2226.3001.4187)
2. [机器学习(六)：朴素贝叶斯及拉普拉斯修正-CSDN博客](https://blog.csdn.net/qq_42103091/article/details/122405255?ops_request_misc=%7B%22request%5Fid%22%3A%2209ec2d59a4414585e8d4f9361c8cba3b%22%2C%22scm%22%3A%2220140713.130102334.pc%5Fall.%22%7D&request_id=09ec2d59a4414585e8d4f9361c8cba3b&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_ecpm_v1~rank_v31_ecpm-2-122405255-null-null.142^v100^pc_search_result_base4&utm_term=%E5%9F%BA%E4%BA%8E%E6%8B%89%E6%99%AE%E6%8B%89%E6%96%AF%E5%8F%98%E6%8D%A2%E7%9A%84%E8%B4%9D%E5%8F%B6%E6%96%AF&spm=1018.2226.3001.4187)
3. [朴素贝叶斯（二）模型、推导、拉普拉斯平滑\_朴素贝叶斯拉普拉斯变换-CSDN博客](https://blog.csdn.net/xiaoxiaoliluo917/article/details/103037466?ops_request_misc=%7B%22request%5Fid%22%3A%2209ec2d59a4414585e8d4f9361c8cba3b%22%2C%22scm%22%3A%2220140713.130102334.pc%5Fall.%22%7D&request_id=09ec2d59a4414585e8d4f9361c8cba3b&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_ecpm_v1~rank_v31_ecpm-7-103037466-null-null.142^v100^pc_search_result_base4&utm_term=%E5%9F%BA%E4%BA%8E%E6%8B%89%E6%99%AE%E6%8B%89%E6%96%AF%E5%8F%98%E6%8D%A2%E7%9A%84%E8%B4%9D%E5%8F%B6%E6%96%AF&spm=1018.2226.3001.4187)

# 基本任务1.2

# 问题理解与分析

编程实现 k 近邻分类器，在西瓜数据集 3.0a 上比较其分类边界与决策树

分类器边界之异同。

由于KNN需要指定K值才能进行训练，而具体的K值应该如何取值是未知的，所以我分别令K=1，K=3，K=5进行了不同实验，得到了不同的分类结果。同时利用绘图包进行了决策边界可视化。再利用相同数据集，利用采用基尼指数的决策树模型进行训练，能够得到不同的分类结果以及决策边界，将其与KNN模型的训练结果进行比较分析，从而得到结论。

# 算法原理阐述

## 2.1 KNN算法原理

k近邻（k-Nearest Neighbor，简称kNN）学习是一种常用的监督学习方法，其工作机制非常简单：给定测试样本，基于某种距离度量找出训练集中与其最靠近的k个训练样本，然后基于这k个“邻居”的信息来进行预测。通常，在分类任务中可使用“投票法”，即选择这k个样本中出现最多的类别标记作为预测结果；在回归任务中可使用“平均法”，即将这k个样本的实值输出标记的平均值作为预测结果；还可基于距离远近进行加权平均或加权投票，距离越近的样本权重越大。与前面介绍的学习方法相比，k近邻学习有一个明显的不同之处：它似乎没有显式的训练过程。事实上，它是“懒惰学习”（lazy learning）的著名代表，此类学习技术在训练阶段仅仅是把样本保存起来，训练时间开销为零，待收到测试样本后再进行处理；相应的，那些在训练阶段就对样本进行学习处理的方法，称为“急切学习”（eager learning）。

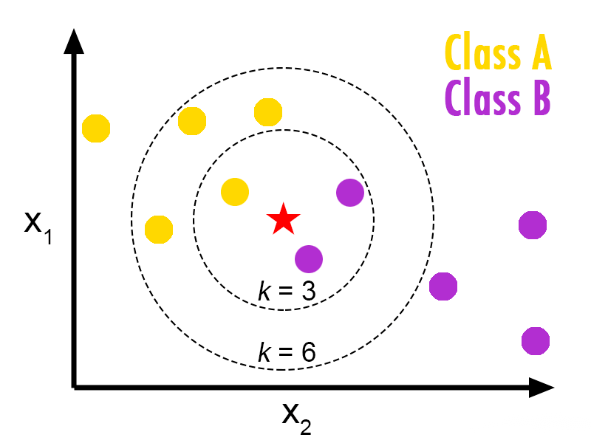


图 4 KNN原理图

如上图所示，当kNN的k定义为3，则在五角星最近的3个点内，分类为B的点最多，则五角星的分类为B；当kNN的k定义为6，则在五角星最近的6个点内，分类为A的点最多，则五角星的分类为A。显然，当k取不同值时，分类结果会有显著不同。另一方面，若采用不同的距离计算方式，则找出的“近邻”可能有显著差别，从而也会导致分类结果有显著不同。

## 2.2 决策树算法原理

决策树是一种常见的分类与回归方法，采用树结构来表示决策过程。决策树由根节点、内部节点和叶节点组成。每个非叶节点代表一个特征属性的测试，每个分支则对应于该特征属性在某个值域上的输出，而每个叶节点存储一个类别或数值。

### 2.2.1 树结构概述

根节点：代表所有样本的起始点。

内部节点：表示对特征属性的测试。

叶节点：表示决策结果或输出值。

从根节点到每个叶节点的路径代表了一个决策的过程，每一步的决策都是基于当前节点的特征属性进行的。最终到达的叶节点就是分类结果或预测值。

### 2.2.2 决策树构建过程

构建决策树的核心在于选择最优特征进行划分，常用的选择标准包括信息增益、信息增益比和基尼指数等，这三种选择标准的算法可参见下文的原理介绍。

决策树学习的基本算法如下：

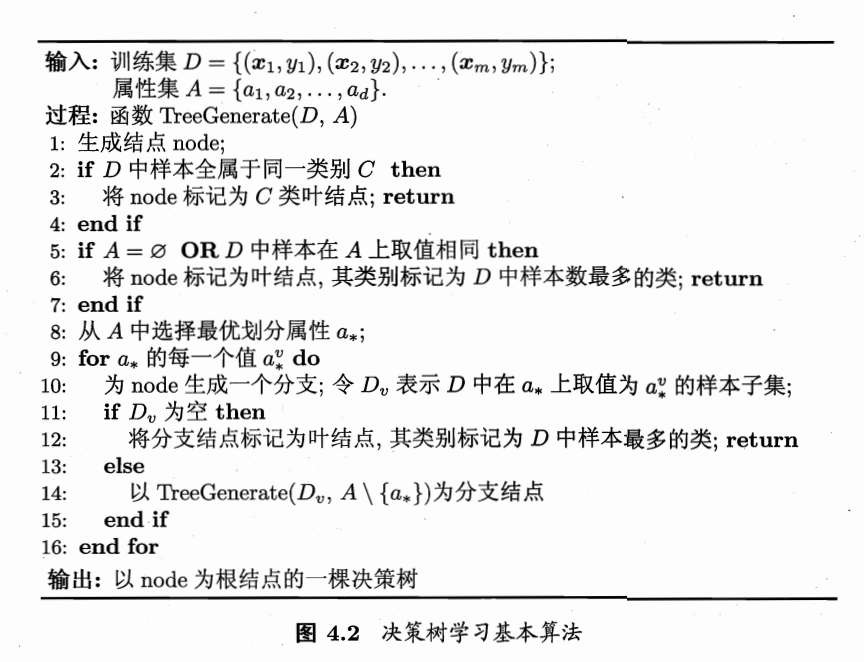


图 5 决策树学习基本算法

决策树的生成是一个递归过程，其递归的终止条件一般有：

（1）所有样本属于同一类别：如果当前节点的所有样本都属于同一类别，则该节点成为叶子节点，并标记为该类别。

（2）没有特征可供选择：如果当前节点没有特征可供选择（所有特征都已使用），则该节点成为叶子节点，并标记为当前节点中样本数最多的类别。

（3）达到预设的深度：如果决策树的深度达到了预设的最大深度，则停止递归。

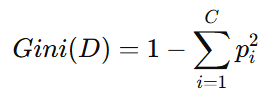
（4）样本数过少：如果当前节点的样本数少于某个预设的阈值，则停止递归。

## 2.2 基尼指数算法原理阐述

在机器学习中，基尼系数常用于决策树算法，尤其是CART（Classification and Regression Trees）算法中，作为特征选择的标准。

#### 1.基尼不纯度

基尼系数被用来衡量一个数据集的纯度。在分类问题中，基尼不纯度定义为：

​

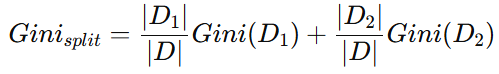
其中，pi是数据集中属于类别i的样本比例，C是类别的总数。基尼不纯度越小，表示数据集越纯，即同一类别的样本越集中。

#### 2.特征选择

在构建决策树时，算法会评估每个特征的基尼不纯度，以决定如何划分数据。选择使得子集基尼不纯度最小的特征进行分裂，从而提高模型的预测准确性。

划分数据集

假设特征 X将数据集 划分为两个子集 和 ，则划分的基尼不纯度计算为：



其中，|| 是数据集中样本的总数，|| 和 || 是划分后子集的样本数。

#### 3.最优划分

通过计算所有特征的基尼不纯度，算法选择基尼不纯度最小的特征进行划分，反复进行，直到达到停止条件（如树的深度限制或节点样本数小于某个阈值）。

# 算法设计思路

KNN（K-Nearest Neighbors，K近邻算法）是一种常用的监督学习算法。其基本思想是通过计算样本点与训练集样本之间的距离，选择离待分类点最近的K个点，根据这些点的标签来确定待分类点的类别。在本实验中，使用KNN算法对西瓜数据集进行分类预测，并通过可视化结果呈现分类边界。

首先设计一个knn函数，传入参数分别为训练集数据、测试集数据以及K值大小，在函数中完成knn模型的实现，设置类别划分阈值为0.5，即大于0.5时，划分为正类；小于0.5时，划分为负类。然后针对不同的K值分别进行训练，将训练结果以决策边界的形式可视化，从图中能够直观地看出划分的正确率。

决策树模型，采用基尼指数进行划分，并利用sklearn自带的算法进行实现，然后输出决策树结果以及决策边界可视化图，并与KNN模型进行比较分析。

# 数据预处理

数据预处理是机器学习中至关重要的一步，旨在将数据转化为适合模型训练的格式。

本实验中，首先读取西瓜数据集 3.0a.csv文件，并将其转换为 numpy 数组以便进行后续处理。数据集包括多个特征列，最后一列为标签（好瓜、坏瓜）。在 knn 函数中，使用 train[j, :-1] 来获取每个训练样本的特征，train[i[0], -1] 用于获取标签列。

# 具体实现

## 5.1 代码结构

主要代码结构可以分为以下几个部分：

（1）KNN分类器的实现：

knn 函数实现了 KNN 算法的基本流程。对于每个测试样本，计算其与所有训练样本的欧氏距离，然后选取距离最小的 K 个训练样本，最后根据这些样本的标签来决定测试样本的标签。

def knn(train, test, num):

output = [] # 存储预测结果

m, n = len(train), len(test)

for i in range(n):

dist\_ij = []

for j in range(m):

d = np.linalg.norm(test[i, :] - train[j, :-1]) # 欧氏距离计算

dist\_ij.append((j, d))

id\_min = sorted(dist\_ij, key=lambda x: x[1])[:num] # 选取K个最近的样本

rate = [train[i[0], -1] for i in id\_min] # 获取K个近邻的标签

if sum(rate) / num >= 0.5:

output.append(1)

else:

output.append(0)

return output

（2）生成网格点： 通过 np.meshgrid 生成一个二维网格，用于在整个特征空间内进行分类预测。这里设定了密度 (x) 和含糖率 (y) 的取值范围，并构造了一个二维网格。

a = np.arange(0, 1.01, 0.01)

b = np.arange(0, 0.61, 0.01)

x, y = np.meshgrid(a, b)

（3）进行分类预测： 使用 KNN 算法对每一个网格点进行预测，并将结果保存到 z 中。之后，将预测结果 z 通过 reshape 转换为与网格相同形状的二维数组，以便于可视化。

z = knn(data, np.c\_[x.ravel(), y.ravel()], k)

z = np.array(z).reshape(x.shape)

（4）结果可视化： 使用 matplotlib 库绘制分类结果的可视化图。通过 ax.contourf 绘制分类边界，并使用 ax.scatter 绘制训练样本的分布。

fig, ax = plt.subplots(1, 1, figsize=(5, 4))

ax.contourf(x, y, z, cmap=plt.cm.winter, alpha=.6)

ax.scatter(data[:, 0], data[:, 1], c=data[:, 2], cmap=plt.cm.PuBuGn, edgecolors='k')

ax.set\_xlim(0, 1.0)

ax.set\_ylim(0, 0.6)

ax.set\_ylabel('含糖率')

ax.set\_xlabel('密度')

ax.set\_title(' %s-近邻分类器' % k)

plt.show()

## 5.2 实验流程

首先，从 CSV 文件中加载西瓜数据集，并使用 pandas 库将其转换为 numpy 数组。接着，将数据中的特征和标签分离，以便后续进行分类操作。然后，生成一个二维网格，覆盖特征空间中的所有可能点，为后续的分类预测提供基础。接下来，利用 KNN 算法对网格中的每个点进行分类预测。具体来说，对于每个待分类点，计算它与训练样本之间的欧氏距离，选取 K 个最近的训练样本，并根据这些邻居的标签决定待分类点的类别。最后，通过 matplotlib 库将分类结果进行可视化，绘制分类边界图并显示训练数据点的分布，从而直观展示 KNN 分类器在不同特征空间下的表现。

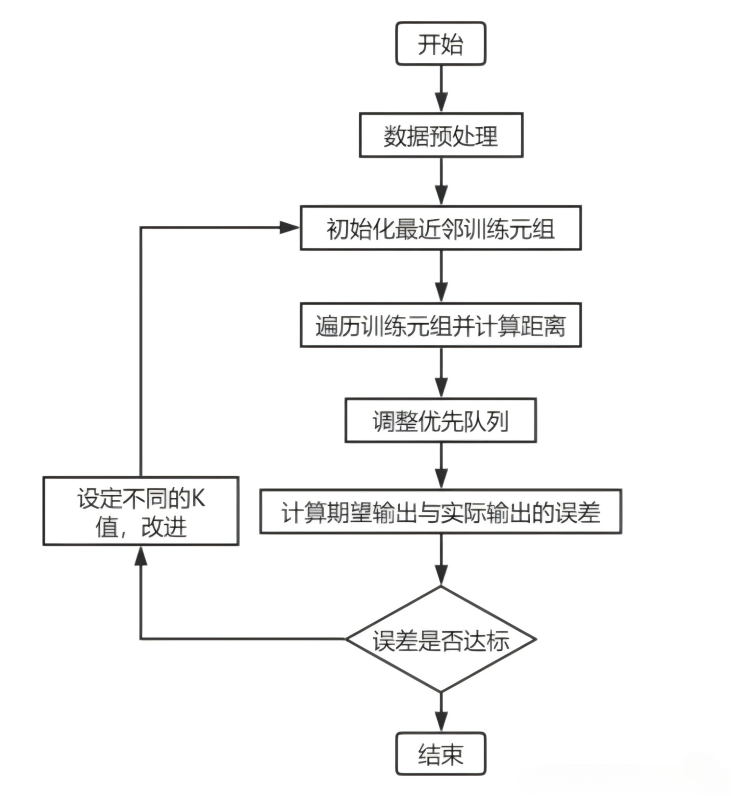


图 6 实验流程图

# 实验结果分析

## 6.1 结果可视化

首先对西瓜数据集3.0a进行可视化，如下图。

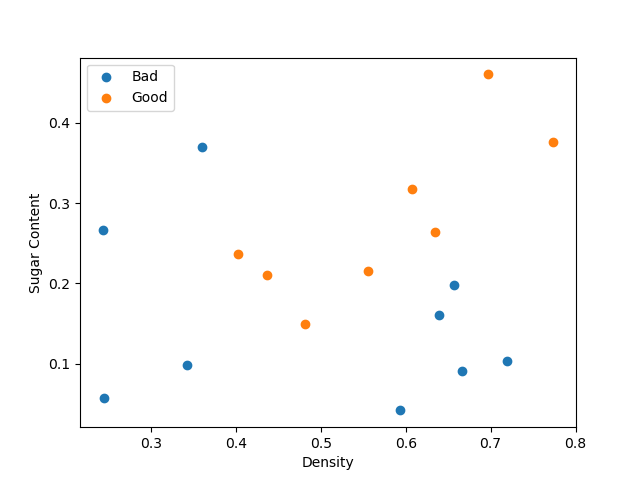


图 7 数据可视化

从数据可视化图可以看到，好瓜与坏瓜数量较平衡，且数据分布较分散，说明含糖了与密度的规律性并不强。

## 6.2 KNN模型

分别令k=1,3,5，基于KNN模型进行模型训练，得到不同的训练边界。其中白色圆点为好瓜，绿色圆点为坏瓜。当k=1时，17个样本预测正确17个，正确率为100%；当k=3时，17个样本预测正确17个，正确率为100%；当k=5时，17个样本预测正确15个，正确率为88.2%。

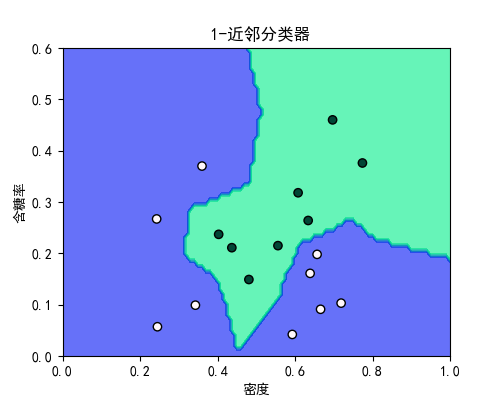


图 8 k=1,KNN决策边界

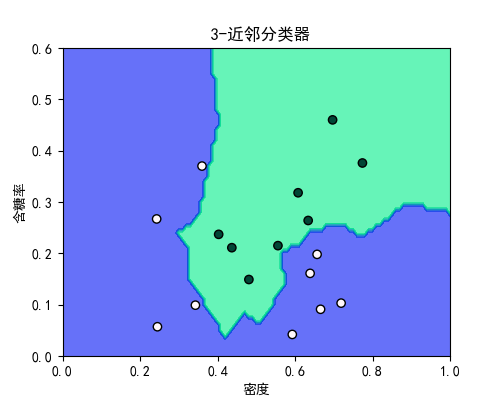


图 9 k=3,KNN决策边界

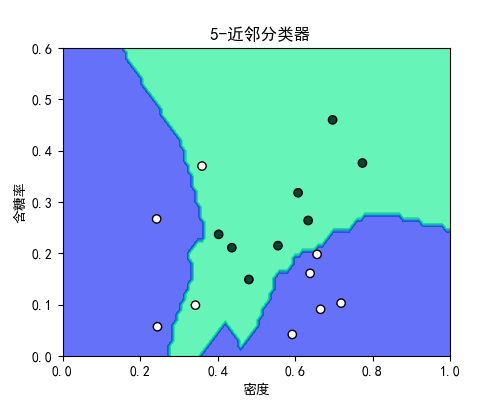


图 10 k=5,KNN决策边界

分析以上三张KNN决策边界图可以看到，K=1与K=3、K=5时的决策边界。区别都比较大，但K=1与K=3时的准确率均为100%，精准的区别了好瓜与坏瓜，相比之下，K=5时的决策边界并没有那么准确，将两个好瓜误判成了坏瓜。整体上看，K=5时的坏瓜决策边界面积更大，模型更趋向于将数据预测为坏瓜。这可能是因为，由于数据量较小，而K=5对于小数据量来说范围较大，所以预测不够精确，导致模型更倾向于将数据预测为坏瓜。而K=3时的决策边界相比较与K=1时，K=3的决策边界不够鲜明，因为边界周围就有数据，而K=1时的边界面积较小，当样本量增加时，可能泛化性会低于K=3。

## 6.3 决策树模型

同样的数据集，再次利用决策树进行训练。作为对比，决策树17个样本预测正确15个，正确率为88.2%。当k值选择合理时，kNN算法的性能要优于决策树。

决策树得到的划分结果如下：

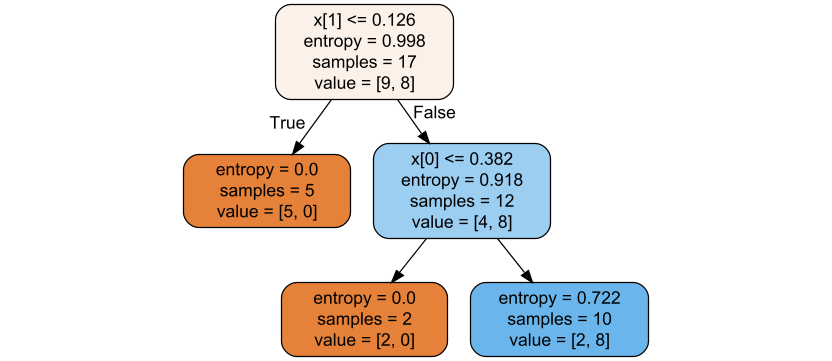


图 11 决策树划分结果

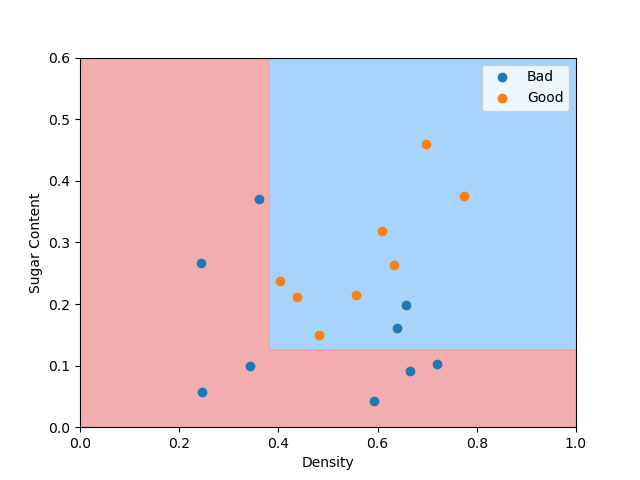


图 12 决策树模型决策边界

决策数模型得到的决策边界成矩形，不同与KNN模型的不规则形状。而现实实际中数据的分布往往呈现出不规律性，过于方正的决策边界显然不适合做判别。故认为当k值选择合理时，kNN算法在西瓜数据集 3.0a 上的性能要优于决策树。

## 6.4 KNN模型与决策树模型的比较

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **模型** | **准确率** | **决策边界** | **优点** | **缺点** |
| KNN (K=1) | 100% | 精确且局部化的边界 | 高精度，适应小数据集 | 可能过拟合，泛化能力较差 |
| KNN (K=3) | 100% | 平滑的决策边界 | 泛化能力较好，适应性强 | 较为平缓的边界可能不适应复杂数据 |
| KNN (K=5) | 88.2% | 模糊的决策边界 | 更强的泛化能力 | 分类精度降低，边界不精确 |
| 决策树 | 88.2% | 矩形边界 | 易于理解，处理复杂的非线性问题 | 划分过于规则，不适应不规则数据 |

# 解决问题主要收获

## 7.1 解决的问题

在决策树模型的构建和分析过程中，如何直观地展示决策树的结构和决策过程是一个关键问题。虽然决策树的规则性使其易于理解，但当树结构较复杂时，通过文字描述或代码展示往往不够清晰。为了有效地展示决策树的结构，我尝试使用 graphviz 进行可视化，但是可视化的过程中遇到了种种困难。后续我参考网上博客，使用 graphviz 库将决策树的分支结构和决策节点以图形的方式展现出来，这样不仅能帮助更好地理解决策树的决策过程，还能直观地看到各个特征如何影响最终的分类结果。

## 7.2 主要收获

通过本次实验，我深入理解了 KNN 算法的原理及应用，通过计算待预测样本与训练数据之间的欧氏距离，选取 K 个最近邻样本，根据邻居标签进行预测。通过分别设置不同的 K 值（K=1、K=3、K=5），观察了决策边界的变化以及模型预测准确率的差异。发现当 K 值较小时（如 K=1），模型能够精确地拟合训练数据，但可能出现过拟合现象；而当 K 值增大时（如 K=5），模型的泛化能力更强，但决策边界可能变得不够精确。这让我认识到 K 值的选择直接影响分类模型的性能，过小的 K 值容易过拟合，过大的 K 值则可能导致欠拟合。此外，通过与决策树算法的对比，我认识到 KNN 在数据分布较为复杂且不规则时表现更佳，而决策树的规则性边界则可能导致分类效果不理想。实验中的可视化分析帮助我更直观地理解了决策边界的变化和模型性能，进一步加深了我对模型调优、参数选择以及算法应用的理解，为将来选择合适的机器学习模型提供了宝贵的经验。

# 参考来源

1. [K-近邻算法： k-nearest neighbor classification (kNN) 详细介绍\_k近邻-CSDN博客](https://blog.csdn.net/chenhepg/article/details/105409153)
2. [KNN算法（k近邻算法）原理及总结-CSDN博客](https://blog.csdn.net/m0_74405427/article/details/133714384)
3. [决策树可视化【Matplotlib/Graphviz】\_决策树可视化graphviz-CSDN博客](https://blog.csdn.net/shebao3333/article/details/105330167)

# 综合任务二

# 一、问题理解与分析

## 1.1 问题背景

### 1.1.1 信用卡欺诈检测的背景和重要性

随着电子支付的普及，信用卡已成为全球最广泛使用的支付工具之一，尤其是在电子商务和在线支付领域。根据统计数据，全球信用卡交易的金额和频次正在以惊人的速度增长。然而，与此同时，信用卡欺诈行为也逐渐增多，给金融机构、商家以及消费者带来了巨大的经济损失。信用卡欺诈不仅直接造成财务损失，还影响到客户对金融服务的信任度，进而损害金融机构的品牌声誉。

信用卡欺诈行为具有以下特点：

1. 隐蔽性强：欺诈者通常通过伪装、盗取信息或滥用支付渠道，伪造交易，导致欺诈行为极具隐蔽性，很难被传统规则检测到。
2. 少数类问题：在大量正常交易数据中，欺诈交易仅占极小比例。这种不平衡的数据分布使得机器学习模型难以识别少数类（即欺诈交易）。
3. 多样化与变化性：随着技术的发展，欺诈手段也在不断变化，新的欺诈模式和手法不断涌现，给欺诈检测系统带来了巨大的挑战。

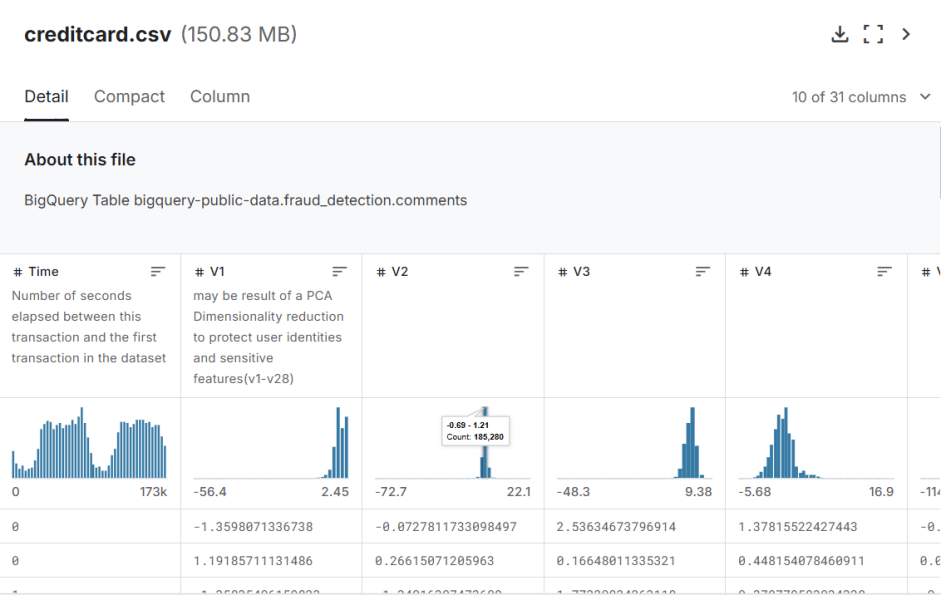
在这种背景下，如何快速、准确地识别信用卡欺诈交易成为了金融行业亟待解决的重要问题。传统的信用卡欺诈检测方法多依赖于规则和人工审核，但这些方法存在以下局限性：

1. 实时性差：人工审核和基于规则的检测方法通常无法实时处理海量交易数据，导致欺诈交易可能在交易完成后被发现，造成较大的经济损失。
2. 准确性低：传统方法的准确率受限于规则设计的复杂性和人工经验，难以适应复杂且多变的欺诈行为。
3. 灵活性差：随着新的欺诈手段不断涌现，传统方法难以进行及时的更新和调整，导致其检测效果逐渐降低。

因此，采用数据驱动的机器学习方法，结合先进的算法和模型来自动化检测欺诈行为，已经成为金融机构提升欺诈检测效果和实时响应能力的关键解决方案。

### 1.1.2 数据集介绍

本实验使用的数据集通常是公开的信用卡交易数据集，最常见的是 Kaggle 上的 **Credit Card Fraud Detection 数据集**，该数据集包含了大量的信用卡交易记录，并已标注出哪些交易是欺诈行为。



该数据集的详细介绍如下：

#### 1.数据来源与结构：

数据集来源于Kaggle网站（https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud），包含信用卡交易的历史记录，数据项包括交易的时间、金额、交易类型、以及其它匿名化处理的交易特征。

#### 2.主要特征与标签：

数据集包含30个匿名的特征（V1到V28），这些特征是通过主成分分析（PCA）从原始交易数据中提取的。还有几个额外的特征：Time（交易时间）、Amount（交易金额）。

Class列标记了每笔交易是否为欺诈交易。Class取值为1表示该交易为欺诈，取值为0表示该交易为正常交易。

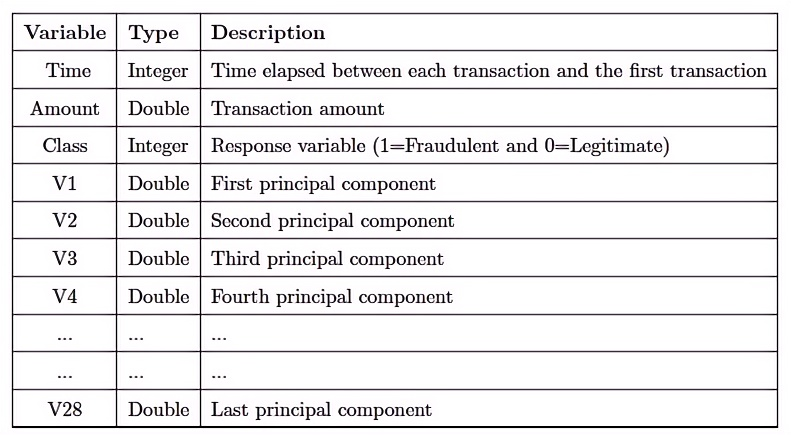


图 13 数据集特征描述

#### 3.匿名化处理与特征选择：

数据集中的V1到V28特征是通过PCA对原始数据进行降维和匿名化处理后得到的，因此并没有显式地标明每个特征的具体含义。此举保护了用户隐私，但同时也使得特征的解释变得困难。

#### 4.数据集的大小与维度：

样本总数：数据集包含约 284,807 条交易记录。

类别数量：数据集的 Class 列包含两个类别：

正常交易（0），占比大约 99.83%。

欺诈交易（1），占比大约 0.17%，属于典型的不平衡数据问题。

属性特征维度：数据集共有 31个属性（30个匿名化特征 + Time 和 Amount）。

特征维度较高，可能需要采用降维或特征选择方法来提高模型的训练效率。

文件大小：数据集文件大小约为 20MB，存储在CSV格式中，适合进行机器学习模型的训练和测试。

该数据集由于其高度不平衡的类别分布，成为了机器学习领域研究不平衡数据问题和欺诈检测的经典案例。

## 1.2 目标与挑战

本实验的目标是通过数据预处理以及多个机器学习模型的训练，找出表现最好、最高效的一个的机器学习模型，用于检测信用卡交易中的欺诈行为。

然而，在信用卡欺诈检测（Credit Card Fraud Detection）数据集中，实现高效欺诈检测面临多重挑战。首先，类别不平衡问题尤为突出，正常交易样本数量远远超过欺诈交易样本，导致数据集呈现极端不平衡分布。这种不平衡使得传统分类算法倾向于将大部分样本预测为正常交易，从而严重降低了欺诈交易的检测率。其次，数据集通常具有高维度特征，例如包含交易时间、金额、地理位置等多种信息，随着特征数量的增加，模型可能陷入“维度灾难”，即数据稀疏性和计算复杂度显著上升，这不仅影响模型的训练效率，还可能导致过拟合现象。此外，信用卡欺诈检测数据集通常规模庞大，包含数百万甚至更多的交易记录，处理如此大规模的数据对计算资源和算法效率提出了极高要求。综上所述，类别不平衡、高维度数据以及数据量庞大是信用卡欺诈检测领域亟待解决的核心挑战。

# 算法原理阐述

课内三种模型的算法原理可参见报告前半部分，这里主要介绍运用到的集中课外模型。

## 2.1 神经网络（Neural Network）

神经网络是一种模仿生物神经系统的信息处理模型，通过构建层层神经元的网络结构来学习数据间的复杂非线性关系，其主要应用于分类、回归、降维等任务中，尤其在处理高维、非线性和复杂模式的场景下表现出色。一个典型的神经网络由输入层、隐藏层（可以有多个）和输出层组成，其中每一层由若干个神经元构成。每个神经元是一个计算单元，其输入来自上一层神经元的输出，经过加权求和后，通过激活函数产生输出，从而完成信息的传递。

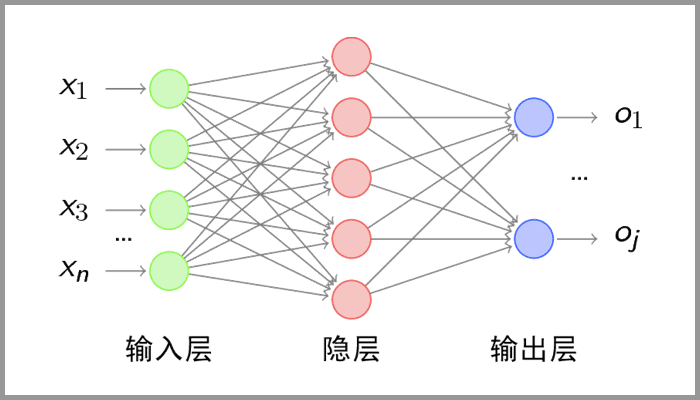


图 14 神经网络原理图

对于神经网络中的一个神经元，其输入和输出关系可以表示为以下数学形式：



其中，表示第层第个神经元的加权输入，表示第层中连接上一层第个神经元与当前层第个神经元的权重，表示偏置，为上一层第个神经元的输出或激活值，是上一层的神经元数量。

随后，神经元的输出通过激活函数计算得到，公式如下：



神经网络的信息处理分为前向传播（Forward Propagation）和反向传播（Backpropagation）两个阶段。在前向传播中，数据从输入层逐层通过隐藏层传递至输出层，每层的计算均按照上述公式进行。前向传播的整体公式为：



其中，和分别表示隐藏层和输出层的权重矩阵，和为相应的偏置项，表示激活函数。

为了引入非线性特性，激活函数是神经网络中的关键组成部分，常用的激活函数包括：

Sigmoid 函数：

Tanh 函数：

ReLU 函数：

神经网络的训练目标是通过最小化损失函数，使预测值接近真实值。损失函数的选择依据任务类型而定，例如，回归问题中常用均方误差（Mean Squared Error, MSE），公式为：



对于分类问题，通常使用交叉熵损失（Cross-Entropy Loss）：



损失函数的优化依赖于反向传播算法，通过链式法则逐层计算梯度。以第层神经元为例，反向传播的核心公式如下：

1. 损失对输出的梯度：



1. 损失对权重的梯度：

​

通过计算上述梯度，权重和偏置更新的公式为：



其中，为学习率，决定了参数更新的步长。

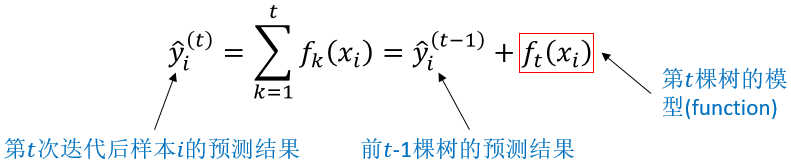
神经网络的优点在于其强大的拟合能力，能够处理复杂的非线性关系，并在语音识别、图像分类等任务中表现优异。但其缺点也较为明显：训练时间较长，依赖大量计算资源，并且容易过拟合，需借助正则化方法（如 L2 正则化和 Dropout）进行优化。

## 2.2 XGBoost

XGBoost算法的核心是采用集成思想——Boosting思想，将多个弱学习器通过一定的方法整合为一个强学习器。它采用前向加法模型，用多棵树共同决策，每棵树的结果都是目标值与之前所有树的预测结果之差，并将所有的结果累加即得到最终的结果。这样可以大大提高整个模型的效果。

### 2.2.1 XGBoost 预测原理

XGBoost是由k个基模型组成的一个加法模型，假设我们第t次迭代要训练的树模型是则有：



### 2.2.2 XGBoost 的目标函数

为了优化每棵树的结构，XGBoost 使用了正则化目标函数，定义为：



其中：

是损失函数，衡量真实值和预测值的差异（常用平方误差或对数损失），是正则化项，用于控制模型的复杂度，定义为：



​

T是树的叶子节点数量，

是第个叶子节点的权重，

和是超参数，用于调节模型的复杂度。

### 2.2.3 XGBoost的优化过程

XGBoost采用前向分步优化 (Greedy Forward Additive Training)。在第次迭代中，假设当前模型的预测值为：

,

则第棵树的优化目标为：



使用二阶泰勒展开对目标函数进行近似，得：

,

其中：

为一阶梯度，

为二阶梯度。

### 2.2.4 树的分裂增益

XGBoost 通过评估分裂点的增益来选择最优的分裂。分裂增益定义为：



其中：

和是分裂后左子节点和右子节点的一阶梯度之和，

和是分裂后左子节点和右子节点的二阶梯度之和，

是正则化参数，

是叶子节点分裂的惩罚参数。

### 2.2.5 叶子节点的权重计算

对于每个叶子节点，最优的权重计算公式为：



其中：

G是该叶子节点的一阶梯度之和，

H是该叶子节点的二阶梯度之和。

## 2.3 LightGBM

LightGBM (Light Gradient Boosting Machine) 是一种基于梯度提升决策树 (Gradient Boosting Decision Tree, GBDT) 的高效实现，它通过优化传统 GBDT 的训练方式来提升模型效率和性能，尤其在处理大规模数据时表现出色。其核心特点包括直方图算法、基于叶子增长的策略、高效的数据分裂方法以及对稀疏数据的支持。

### 2.3.1 LightGBM 预测原理

LightGBM 是由k个弱学习器（决策树）组成的加法模型，其预测值表示为：

,

其中：

为第i个样本的预测值，

是第t棵决策树对样本的预测结果，

是所有树模型的函数空间。

LightGBM通过逐步优化目标函数来训练每棵树，使预测结果不断逼近真实值。

### 2.3.2 LightGBM 的目标函数

LightGBM的目标函数与XGBoost类似，由损失函数和正则化项组成：



其中：

是损失函数，衡量真实值和预测值之间的差异（例如平方误差或对数损失，是正则化项，用于控制树模型的复杂度，定义为：



其中T为叶子节点数，为叶子节点权重，和为正则化参数。

### 2.3.3 核心优化点

LightGBM 采用直方图算法，将特征值离散化为b个等间隔的区间（bin），构造直方图来存储每个区间的梯度和权重统计信息，从而显著减少计算量。特征分裂增益的计算公式为：



其中：

和分别是左子节点和右子节点的一阶梯度之和，

和分别是左子节点和右子节点的二阶梯度之和，

是正则化参数，是叶子分裂惩罚项。

### 2.3.4 叶子节点权重计算

对于每个叶子节点，其最优权重w可通过以下公式计算：



其中：

G 是该叶子节点的一阶梯度之和，

H 是该叶子节点的二阶梯度之和。

### 2.3.5 LightGBM的正则化与控制参数

为了防止过拟合，LightGBM 提供了一系列参数用于控制模型复杂度和训练过程：

：限制树的最大深度，

：限制叶子节点数，

：L2正则化参数，

：叶子分裂惩罚参数。

## 2.4 AutoEncoder

AutoEncoder（自动编码器）是一种典型的无监督学习算法，主要用于学习数据的低维表示。它通过一种编码器-解码器（Encoder-Decoder）的结构，将高维数据映射到一个低维的潜在空间中，并通过解码器从潜在空间中重构原始数据。重构误差的最小化是 AutoEncoder 的优化目标。

### 2.4.1 AutoEncoder 的基本结构

AutoEncoder 的结构由两个主要部分组成：

1. 编码器（Encoder）：将输入数据x映射到一个低维的潜在表示z：



其中：

x是输入数据，

z是潜在表示（Latent Representation），通常维度远小于x，

是编码器函数（例如多层感知机，MLP），由参数θ表示。

1. 解码器（Decoder）：从潜在表示z重构出原始数据：



其中：

是重构后的数据，

是解码器函数（例如MLP），由参数ϕ表示。

最终目标是使重构后的数据尽可能接近输入数据x。

### 2.4.2 AutoEncoder 的目标函数

AutoEncoder 的目标是最小化输入数据和重构数据之间的差异，通常通过定义一个损失函数来实现。常见的损失函数是重构误差，例如均方误差（MSE）：



其中：

n是样本数，

是第i个输入样本，

是第i个样本的重构数据。

通过最小化上述损失函数，模型可以学习到能够良好表征输入数据的潜在表示。

### 2.4.3 编码器与解码器的网络结构

编码器和解码器通常采用对称的神经网络结构,编码器将输入数据x映射到一个低维的潜在空间，通过多层全连接网络逐层压缩数据维度。解码器则通过反向操作将潜在表示z映射回原始数据的高维空间。假设编码器和解码器各有L层，其映射函数可以表示为：





其中：

和是第l层的权重矩阵，

和是偏置项，

是激活函数（例如 ReLU 或 Sigmoid）。

# 三、算法设计思路

## 3.1总体思路

针对信用卡欺诈检测任务，由于数据量确实十分庞大，所以首先上网收集了许多关于不平衡大数据集的处理方法，需要对数据进行多种预处理。首先需要了解数据的基本信息，如是否包含缺失值、重复值等，然后需要对数据进行标准化。

处理完数据之后，结合多种机器学习模型进行实验，本实验采用留一法进行训练集、测试集划分，划分比例为8:2，同时对训练集做采样处理，分为不采样、随机下采样以及SMOTE过采样处理。在不同的采样方式下，分别进行模型训练。本实验采用的课内模型有KNN、基于拉普拉斯变换的贝叶斯模型以及随机森林模型；选用的课外模型有神经网络、XGBoost、LightGBM等模型，并另外使用了一种方法，即首先对数据进行 AutoEncoder 降维训练，得到低维表示；然后，使用t-SNE对得到的低维特征进行进一步降维可视化。最后，使用逻辑回归对处理后的数据进行分类，从而得到了较好的分类效果。

由于数据比例极度不平衡，非欺诈交易（即正常交易）数据占绝大多数，所以不能采用传统的评价指标（如准确率等）进行模型评估。考虑用Precision、Recall、F1-score以及AUC曲线等指标，并绘制混淆矩阵，以全面衡量模型性能。

同时，采用PCA对原数据进行数据降维，得到一份新数据。利用新数据再进行一轮实验，与之前采用原数据进行模型训练得到的结果进行比较分析。

由于模型的参数并不确定，需要不断地进行调参，合适的参数下模型才能够发挥出能好的效果。所以采用超参数调优（如GridSearchCV）对模型进行优化，以提升模型的预测精度以及泛化能力。利用调优后的模型再次进行一轮训练，与之间的训练结果进行比较分析。

## 3.2模型组合与选择

### 3.2.1 课内模型

课内模型我选择了任务一所使用的KNN以及基于拉普拉斯变换的贝叶斯模型，同时还另外选择了随机森林模型。

在选择模型的过程中，我首先考虑到信用卡欺诈检测数据集存在严重的类别不平衡问题，需要一个对此鲁棒的模型，而随机森林通过集成多棵树和随机采样的机制，能够有效缓解这一问题。其次，数据集特征维度较高，随机森林在每棵树中仅使用部分特征，不仅降低了计算复杂度，还能减少冗余特征的影响，非常适合处理高维数据。此外，我模型可能会过拟合，而随机森林通过集成多棵树的结果，天然具有抗过拟合的能力。基于这些考虑，我最终选择了随机森林模型，认为它能够在信用卡欺诈检测任务中提供稳定、可靠且可解释的性能。

### 3.2.2 课外模型

在《机器学习》课程所讲解的基本模型之外，我选择了神经网络、XGBoost、LightGBM等模型，并另外使用了一种方法，即首先对数据进行t-SNE数据降维，然后采用AutoEncoder模型对降维后的数据进行训练，得到处理后的数据，并利用t-SNE进行可视化。最后采用逻辑回归对处理后的数据进行分类，得到了比较好的分类效果。

我选择神经网络是因为它在处理复杂非线性关系方面表现出色，尤其是在信用卡欺诈检测这种高维数据中，神经网络能够自动学习特征之间的复杂交互关系。我希望通过神经网络的强大拟合能力，捕捉到数据中潜在的欺诈模式，从而提升模型的检测性能。

XGBoost是一种高效的梯度提升算法，以其出色的性能和速度著称。我选择它是因为它在处理类别不平衡数据时表现优异，并且能够通过正则化防止过拟合。我希望利用XGBoost的高精度和鲁棒性，在欺诈检测任务中取得更好的分类效果。

LightGBM是基于梯度提升框架的另一种高效算法，特别适合处理大规模数据。我选择它是因为它在训练速度和内存占用方面优于传统方法，同时支持类别不平衡数据的处理。我希望通过LightGBM快速训练模型，并在大规模数据集上实现高效的欺诈检测。

至于最后的AutoEncoder模型与逻辑回归，我选择这种方法是因为t-SNE能够将高维数据降维到低维空间并保留数据的局部结构，而AutoEncoder可以学习数据的潜在表示，进一步提取有用特征。通过t-SNE可视化，可以直观地观察数据的分布情况。最后，使用逻辑回归对处理后的数据进行分类，因为逻辑回归简单且易于解释，能够在降维后的数据上实现较好的分类效果。这种方法帮助我在保留数据关键信息的同时，提升了模型的分类性能。

## 3.3评价指标

在信用卡欺诈检测（Credit Card Fraud Detection）数据集中，由于数据存在严重的类别不平衡问题（正常交易样本远多于欺诈交易样本），传统的准确率（Accuracy）指标容易产生误导，因为即使模型将所有样本预测为正常交易，也能获得较高的准确率。因此，为了全面评估模型的表现，采用以下评价指标：

### 3.3.1 ROC曲线与AUC值

ROC曲线（Receiver Operating Characteristic Curve）通过绘制真正例率（TPR，即召回率）与假正例率（FPR）之间的关系，直观地反映模型在不同阈值下的分类性能。AUC值（Area Under Curve）是ROC曲线下的面积，用于量化模型的整体分类能力。AUC值越接近1，表明模型区分正常交易与欺诈交易的能力越强。

### 3.3.2 分类报告（Classification Report）

分类报告提供了精确率（Precision）、召回率（Recall）、F1得分（F1-Score）等指标的详细结果。其中，召回率（Recall）是欺诈检测中的关键指标，它衡量模型正确识别出的欺诈交易占所有实际欺诈交易的比例。由于欺诈交易样本稀少，召回率的高低直接反映了模型对欺诈行为的检测能力。F1得分是精确率与召回率的调和平均数，能够平衡两者之间的关系，特别适用于类别不平衡的场景。

### 3.3.3 混淆矩阵（Confusion Matrix）

混淆矩阵通过展示真正例（TP）、假正例（FP）、真反例（TN）、假反例（FN）的数量，直观地反映模型的分类结果。在信用卡欺诈检测中，我们重点关注真正例（TP）和假反例（FN），因为假反例表示模型未能识别出的欺诈交易，这对业务的影响尤为严重。

在模型评估过程中，特别关注召回率（Recall）、精确率（Precision）和F1得分，以确保模型能够有效识别欺诈行为。召回率的高值意味着模型能够捕捉到更多的欺诈交易，从而减少漏检的风险；精确率则衡量模型预测为欺诈的交易中实际为欺诈的比例，高精确率能够降低误报率，避免对正常交易的干扰。F1得分是精确率与召回率的调和平均数，能够综合两者的表现，避免了单一指标的片面性。通过结合这些评价指标，我们能够全面评估模型在信用卡欺诈检测中的性能，并针对性地优化模型以应对数据不平衡带来的挑战。例如，在保证较高召回率的同时，通过调整分类阈值或采用集成学习方法，进一步提升精确率，从而实现模型在欺诈检测中的高效性与可靠性。

# 数据预处理

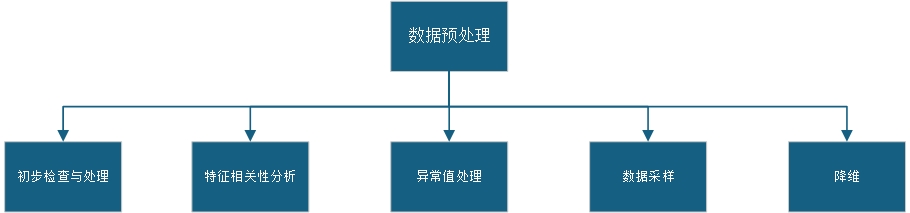
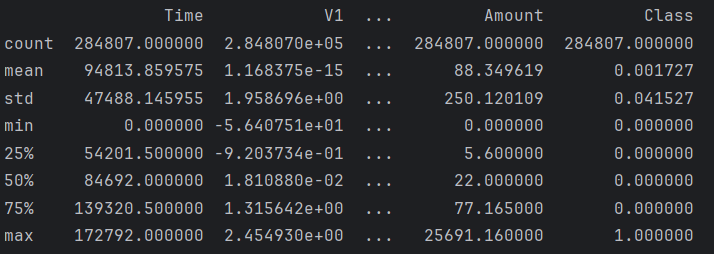


图 15 数据预处理

## 4.1初步检查与处理

首先进行数据集描述、判断是否有空值，发现数据集无空值，得到数据描述如下：



对数据类别的比例进行统计，进行可视化得到如下直方图：

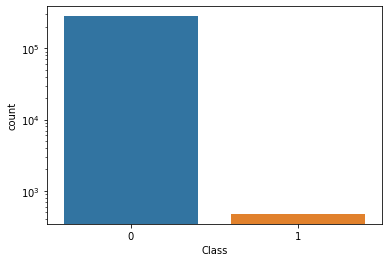


图 16 高度不平衡的类别分布的可视化

通过直方图可以看到，只有492笔（或0.172%）的交易是欺诈的。这意味着数据在目标变量Class方面**高度不平衡**。

由于原数据集中Amount与Time没有经过标准化，故我们首先对其分布进行可视化，随后使用 StandardScaler 对两个变量进行数据标准化，以消除不同特征尺度的影响。

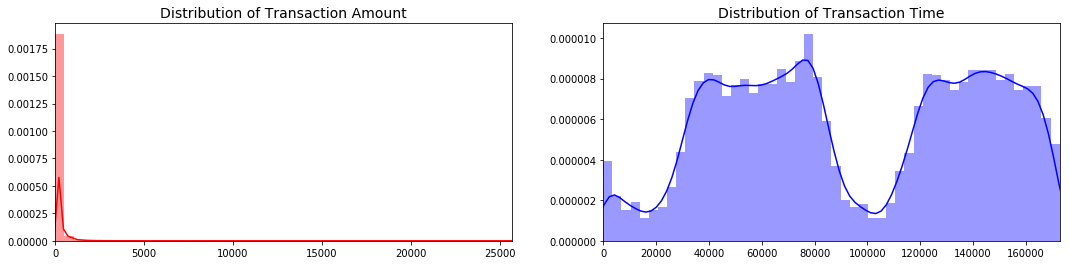


图 17 交易金额与交易时间分布图

观察分布图可以看到，信用卡交易金额呈现右偏分布，即大多数交易金额较小，少数交易金额较大。这有助于识别异常高额交易，可能是潜在的欺诈行为。交易时间的分布可以揭示交易活动的模式，图中可以看的某些时间段的交易量较高。异常时间段的交易（如深夜或凌晨）可能需要进一步检查，以识别潜在的欺诈行为。

## 4.2特征相关性分析

在信用卡欺诈检测任务中，特征工程是提高模型性能的重要环节之一。为了优化模型表现，可以对数据集中的特征进行相关性分析，从而可以知道是否有特征对特定交易是否为欺诈有重大影响。

为了分析特征与目标变量之间的相关性，采用皮尔逊相关系数（Pearson Correlation Coefficient）进行度量，并使用热力图（Heatmap）进行可视化。相关系数的取值范围为[-1, 1]，相关系数接近1表示强正相关，接近-1表示强负相关，接近0则表示无相关性，即这两个特征之间没有线性关系。

首先计算每一对特征与目标变量（欺诈与非欺诈交易）之间的皮尔逊相关系数，得到皮尔逊相关系数矩阵。通过计算得到的相关系数矩阵，可以绘制特征相关性热力图，通过热力图，我们可以直观了解特征之间的相关性，如下图所示。

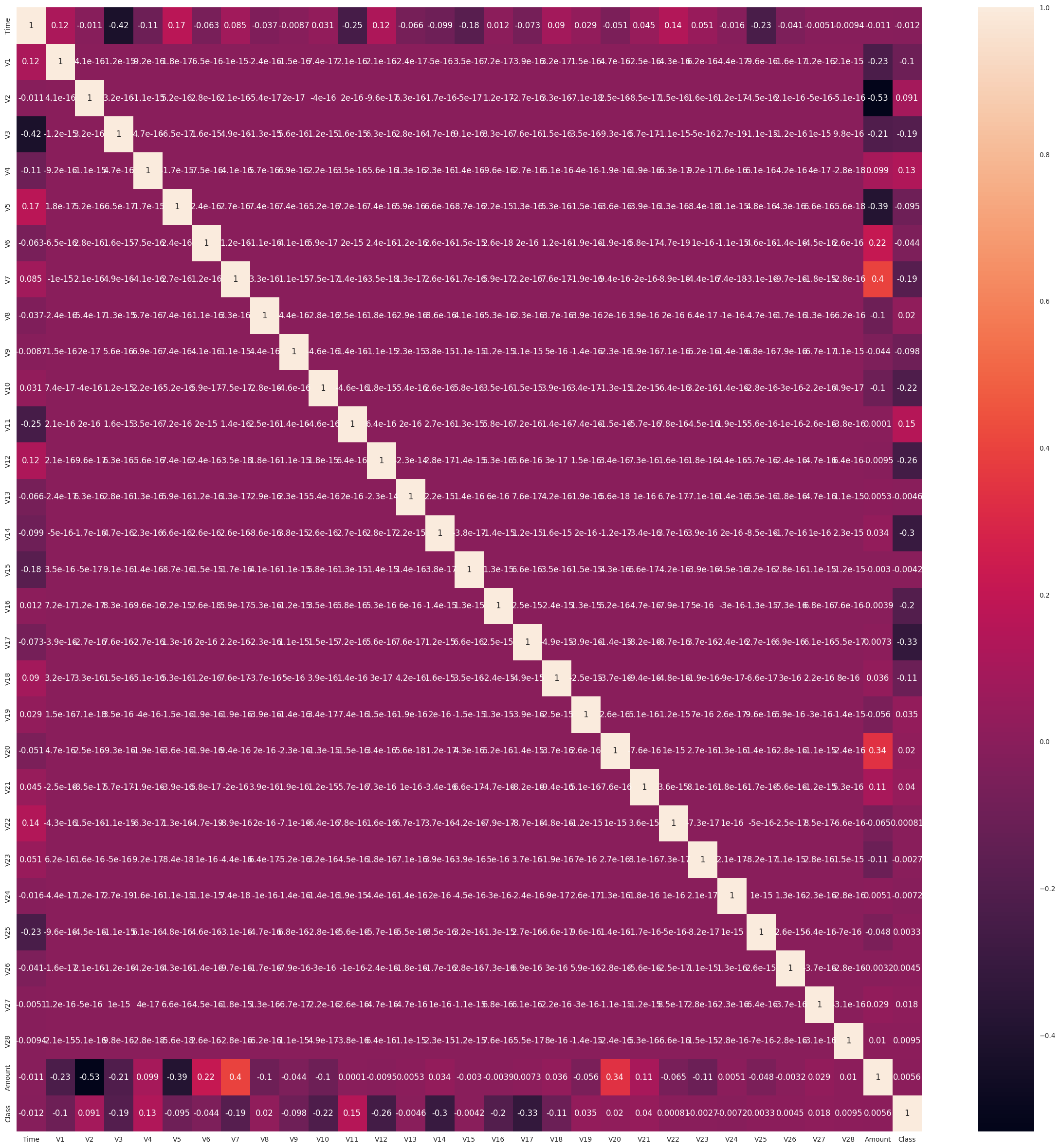


图 18 特征相关热力图

根据热力图的结果，可以发现：

特征V10、V12、V14、V17与class呈较强的负相关。这意味着这些特征的值越低，越可能是欺诈交易。例如，当V10、V12、V14、V17的值较低时，交易更有可能是欺诈。

特征V2、V4、V11和V19与class呈较强的正相关。也就是说，当这些特征的值较高时，交易越可能是欺诈交易。例如，V2、V4、V11 或 V19 的值越高，越倾向于欺诈交易。

通过以上的相关性分析，我们筛选出了与目标变量高度相关的特征，并在下一节中对这些特征进行进一步的异常值检测。异常值检测能够帮助我们识别出数据中的异常模式或错误，从而避免其对模型训练造成负面影响。

## 4.3异常值处理

### 4.3.1 IQR方法的基本原理

IQR（Interquartile Range，四分位距）是一种用于识别和处理异常值的统计方法。其基本原理如下：

1. 计算四分位数：将数据按升序排列，并分为四等份。第一四分位数（Q1）是第25百分位的值，第三四分位数（Q3）是第75百分位的值。
2. 计算IQR：IQR是Q3与Q1的差值，即IQR=Q3−Q1IQR=Q3−Q1。
3. 确定异常值范围：通常，低于Q1−1.5×IQRQ1−1.5×IQR或高于Q3+1.5×IQRQ3+1.5×IQR的值被视为异常值。

IQR方法能够有效识别数据中的极端值，同时避免对数据分布造成过大影响，因此在异常值处理中广泛应用。

### 4.3.2 异常值处理与分析

在本节中，我们对四种较强负相关特征（V17、V14、V12、V10）和四种较强正相关特征（V2、V4、V11、V19）进行了异常值处理，并分析了处理前后的数据分布变化。

处理前的箱线图分析：

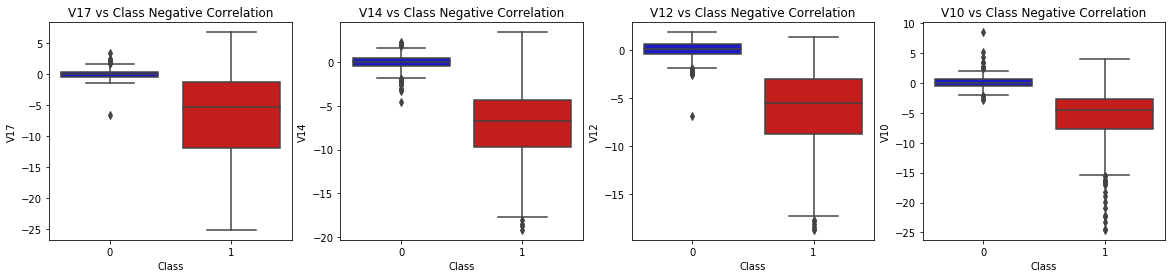


图 19 较强负相关特征箱线图

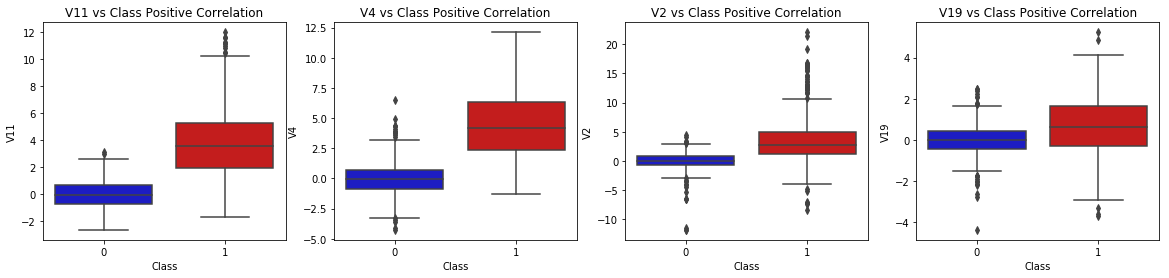


图 20 较强正相关箱线图

图17（较强负相关特征箱线图）：展示了V17、V14、V12和V10在异常值处理前的分布情况。可以看出，这些特征中存在较多的异常值，尤其是极端异常值。

图18（较强正相关箱线图）：展示了V2、V4、V11和V19在异常值处理前的分布情况。同样，这些特征中也存在明显的异常值。

我们使用IQR方法对上述特征进行了异常值处理，重点关注“极端异常值”而非所有异常值。这是因为过度去除异常值可能导致信息丢失，从而降低模型的准确率。

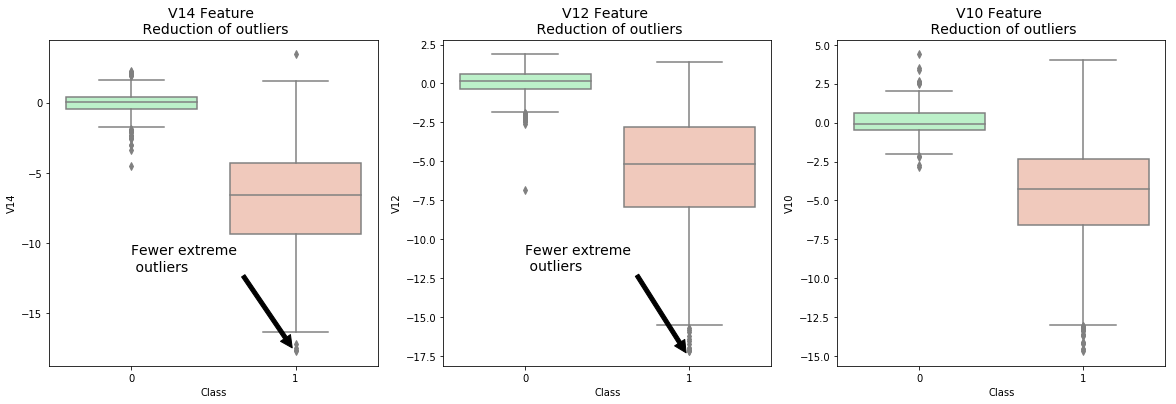


图 21 异常值处理后箱线图

图19（异常值处理后箱线图，以V14为例）：展示了V14特征在异常值处理后的分布情况。可以看出，异常值数量显著减少，数据分布更加集中。

通过IQR方法，我们成功去除了各特征中的极端异常值，同时保留了大部分有效数据。

异常值处理后，数据分布更加合理，减少了噪声对模型训练的干扰，为后续建模提供了更高质量的数据。

## 4.4数据采样

### 4.2.1 随机下采样

当数据集中存在偏差且分布不均时，可以通过采样技术来获得更加均衡的数据分布。在欠采样（Undersampling）方法中，多数类的样本数量会被减少，使其与少数类的样本数量相等。这一过程通常通过随机剔除多数类样本来实现。然而，欠采样的一个主要缺点是可能会丢失一些重要的数据，这些数据可能对欺诈检测模型的性能提升具有关键作用。尽管如此，当数据集规模较大时，减少多数类样本可以显著提高算法的运行效率并缓解存储压力，因此在这种情况下，欠采样是一种较为理想的选择。下图展示了通过欠采样实现数据集平衡的过程。

图 22 随机下采样效果图

### 4.2.2 SMOTE过采样

SMOTE（Synthetic Minority Over-sampling Technique）是一种处理不平衡数据集的过采样技术。它通过生成合成样本来增加少数类样本的数量，进而平衡类别分布。与传统的过采样方法（如复制少数类样本）不同，SMOTE通过在少数类样本之间插值，生成新的样本，从而增加数据的多样性，减少过拟合的风险。

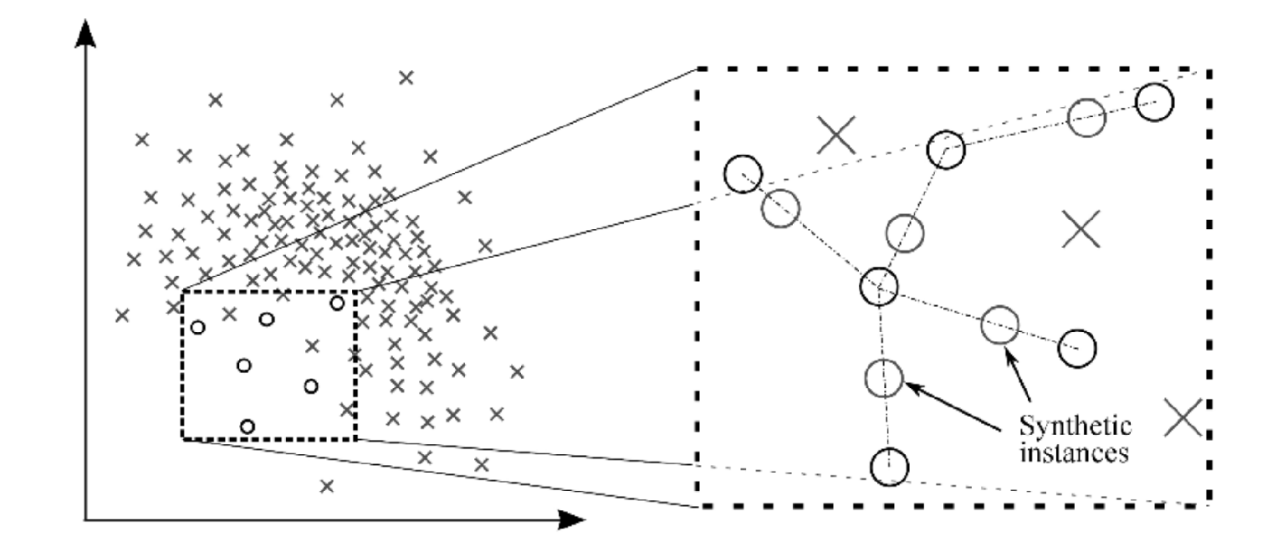


图 23 使用Smote生成合成示例

SMOTE的核心思想是通过对少数类样本之间的距离进行计算，选择距离最近的邻居（通常是k个邻居），然后在这些邻居之间通过线性插值生成新的合成样本。具体步骤如下：

1、对于每一个少数类样本，找到它的k个最近邻。

2、随机选择其中一个邻居，并在原样本与邻居之间生成一个新的样本。

3、重复该过程，直到少数类样本达到所需的数量。

SMOTE的优势在于通过插值生成的合成样本较为多样化，能有效避免过采样时出现的“过拟合”问题。

本实验中，我们将训练集的类别1进行SMOTE进行过采样，以达到数据平衡，便于模型训练，实现效果如下图：

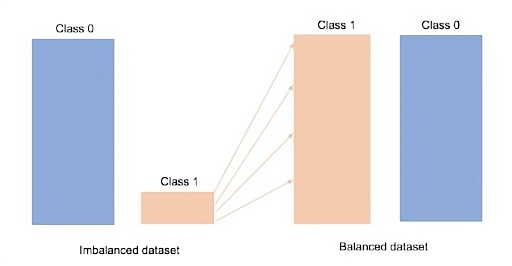


图 24 SMOTE过采样效果图

## 4.5降维

在机器学习任务中，特别是处理高维数据时，特征空间可能非常庞大，导致计算复杂度高，甚至可能会出现维度灾难（Curse of Dimensionality）。为了提升计算效率并减少噪声对模型训练的影响，降维技术被广泛应用。降维可以帮助我们提取最重要的信息，并通过将数据压缩到较低的维度，简化问题的复杂性。本实验采用了两种常用的降维技术——PCA（主成分分析）和t-SNE（t分布随机邻域嵌入）。

### 4.5.1 PCA（主成分分析）

PCA（Principal Component Analysis，主成分分析）是一种经典的线性降维技术，旨在通过将数据投影到一个新的坐标系中，以降低数据的维度，同时尽可能保留数据中的方差信息，从而减少冗余信息对模型的影响。在PCA中，新坐标系的每个轴（即主成分）代表数据中方差最大的方向。通过这种方式，PCA能够有效地压缩数据的维度，使得数据在低维空间中仍然能够保留其最重要的特征。

PCA的工作原理首先要求对原始数据进行标准化处理，使得每个特征具有零均值和单位方差，避免由于不同特征的尺度差异而影响降维结果。接下来，PCA计算数据集中各特征之间的协方差矩阵，协方差矩阵反映了特征之间的相关性。通过对协方差矩阵进行特征值分解，得到特征值和特征向量。特征值代表了主成分在数据中所解释的方差大小，而特征向量则表示主成分的方向。通过选择特征值较大的主成分，可以确保保留数据中方差最大的方向，从而提取出最具代表性的特征。最后，数据通过投影到选定的主成分上，得到降维后的数据。经过PCA降维后的数据，包含了原始数据中最具区分度和最大方差的特征，因此能够提高后续模型的训练效率，降低计算成本，并减少过拟合的风险。

通过对数据集进行PCA，得到一个新的数据集，其中包含了原始数据中方差最大、最有区分力的特征。为了选择合适的主成分数量，我们设置了一个标准，即选择累积方差贡献率超过80%的特征数量。这个标准确保我们保留了数据中绝大多数的信息，同时去除了不重要或冗余的特征，进一步降低了数据的维度。利用这一新的降维后的数据集对模型进行重新训练，与使用原数据进行训练的模型结果进行对比分析。

### 4.5.2 t-SNE（t分布随机邻域嵌入）

与PCA不同，t-SNE（t-distributed Stochastic Neighbor Embedding）是一种非线性降维技术，特别适用于高维数据的可视化。t-SNE的目标是将高维空间中的数据点映射到低维空间，同时尽可能保持数据点之间的局部相似性，使得在低维空间中的数据点能够反映出高维空间中的邻接关系。t-SNE不通过线性方法寻找数据方差的最大方向，而是通过保持数据的局部结构，尽量避免数据点在低维空间中出现过度重叠和挤压。t-SNE的核心思想是通过计算每对数据点在高维空间中的相似度（通常采用高斯分布）以及在低维空间中的相似度（采用t分布），并通过最小化高维空间和低维空间中相似度差异的目标函数，来优化数据的低维表示。

t-SNE特别适用于展示数据的群体结构，并能够有效地揭示数据中各类别之间的关系。在处理具有复杂结构的高维数据时，t-SNE能够保留数据的局部结构和类间差异，因此可以用于探索数据的潜在分布和类别之间的边界。在数据降维后，t-SNE可以将高维数据可视化为二维或三维图，从而帮助我们更直观地理解数据的分布特征，揭示潜在的聚类结构或类别分隔。通过这种方式，t-SNE不仅能够帮助我们降维，而且提供了有效的可视化工具，使得我们能够更好地理解数据的内在关系。

在课外模型的第四种方法，即首先对数据进行t-SNE数据降维，然后采用AutoEncoder模型对降维后的数据进行训练，得到处理后的数据，并利用t-SNE进行可视化。t-SNE起到了巨大作用。

# 具体实现

## 5.1 实验总体设计与流程

首先在一个代码文件中，使用课内模型KNN、基于拉普拉斯变换的贝叶斯模型以及随机森林模型，采用原始数据集，对数据集进行训练集测试集划分后，对训练集分别进行不采样、随机下采样以及SMOTE过采样。对课内的三种模型分别进行训练，得到分类报告、混淆矩阵以及ROC曲线图。并且，对于每个模型，都采用GridSearchCV进行超参数调优，以确保模型得到的结果是最优的。

在另一个代码文件中，使用三种课外模型，分别为神经网络、XGBoost以及LightGBM，同样采用原始数据集，对数据集进行训练集测试集划分后，对训练集分别进行不采样、随机下采样以及SMOTE过采样。与前面的三种课内模型一样，对课外三种模型进行训练，得到分类报告、混淆矩阵以及ROC曲线图。并且对每个模型都进行超参数调优，确保模型得到的结果最优。

另外采用一种方法，首先利用T-SNE降维方法对数据进行处理，将特征数量降维至2，然后进行可视化。然后对数据进行AutoEncoder降维训练，得到低维表示，并使用t-SNE对得到的低维特征进行进一步降维可视化，再次进行可视化后可以发现，两种类别被有效地区分开来。最后利用逻辑回归对训练后的数据进行判别，输出分类报告、混淆矩阵以及ROC曲线图，发现结果较好。

最后将得到的不同训练结果，按照不同的采样方式、模型、数据集进行划分，对比F1分数、Recall值以及Precision、ROC曲线、AUC值等指标，综合评价模型性能。

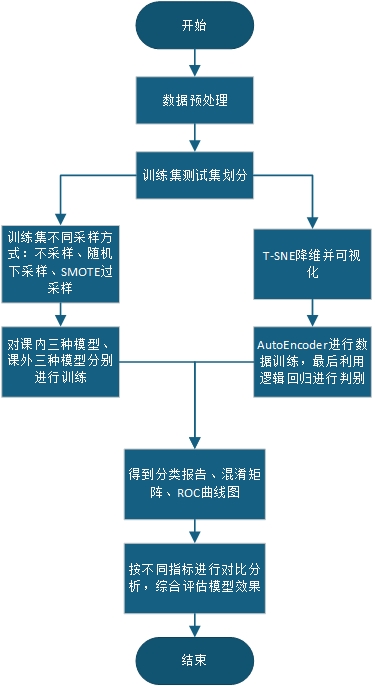


图 25 总体实验流程图

## 5.2 课内模型

### 5.2.1 KNN（K-Nearest Neighbor）：

在本次实验中，使用了KNN模型（k=5）对信用卡欺诈数据进行分类，并分别在不采样、随机下采样和SMOTE过采样三种情况下进行了训练和测试。

得到分类报告及混淆矩阵如下：

表 3 KNN分类报告

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **采样方法** | **类别** | **Precision** | **Recall** | **F1-Score** | **Support** |
| 不采样 | 0 | 1.00 | 1.00 | 1.00 | 56864 |
| 1 | 0.94 | 0.78 | 0.85 | 98 |
| macro avg | 0.97 | 0.89 | 0.92 | 56962 |
| weighted avg | 1.00 | 1.00 | 1.00 | 56962 |
| 随机下采样 | 0 | 1.00 | 0.98 | 0.99 | 56864 |
| 1 | 0.06 | 0.90 | 0.12 | 98 |
| macro avg | 0.53 | 0.94 | 0.55 | 56962 |
| weighted avg | 1.00 | 0.98 | 0.99 | 56962 |
| SMOTE过采样 | 0 | 1.00 | 1.00 | 1.00 | 56864 |
| 1 | 0.48 | 0.87 | 0.62 | 98 |
| macro avg | 0.74 | 0.93 | 0.81 | 56962 |
| weighted avg | 1.00 | 1.00 | 1.00 | 56962 |

|  |  |  |
| --- | --- | --- |
| 课内_KNN 不采样_原_不调优_混淆矩阵 | 课内_KNN 随机下采样_原_不调优_混淆矩阵  图 26 KNN混淆矩阵 | 课内_KNN 过采样_原_不调优_混淆矩阵 |

不进行数据采样的情况下，模型对正常交易（类别0）的分类效果非常好，精确率、召回率和F1分数均为1.00。对欺诈交易（类别1）的召回率为0.78，表明模型能够检测到78%的欺诈交易，但精确率为0.94，说明存在少量误报。

而在随机下采样下，模型对欺诈交易的召回率显著提高（0.90），但精确率大幅下降至0.06，表明模型产生了大量误报。

对数据进行SMOTE过采样后，模型对正常交易的分类效果仍然非常好，加权平均精确率、召回率和F1分数均为1.00。然而，由于数据极度不平衡，我们应该关注宏观平均，过采样下的平均精确率、召回率和F1分数分别为0.74、0.93、0.81，表明，虽然召回率有所上升，但精准率仍然有待提高。

### 5.2.2 基于拉普拉斯变换的贝叶斯分类器

使用了基于拉普拉斯变换的贝叶斯分类器对信用卡欺诈数据进行分类，并分别在不采样、随机下采样和SMOTE过采样三种情况下进行了训练和测试。以下为分类报告及混淆矩阵的结果：

表 4 基于拉普拉斯的贝叶斯模型分类报告

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **采样方法** | **类别** | | **Precision** | **Recall** | | **F1-Score** | **Support** |
| 不采样 | 0 | | 1.00 | 0.99 | | 0.99 | 56864 |
| 1 | | 0.09 | 0.81 | | 0.17 | 98 |
| macro avg | | 0.54 | 0.90 | | 0.58 | 56962 |
| weighted avg | | 1.00 | 0.99 | | 0.99 | 56962 |
| 随机下采样 | 0 | | 1.00 | 0.81 | | 0.90 | 56864 |
| 1 | | 0.01 | 0.91 | | 0.02 | 98 |
| macro avg | | 0.51 | 0.86 | | 0.46 | 56962 |
| weighted avg | | 1.00 | 0.81 | | 0.89 | 56962 |
| SMOTE过采样 | 0 | | 1.00 | 0.99 | | 0.99 | 56864 |
| 1 | | 0.14 | 0.85 | | 0.24 | 98 |
| macro avg | | 0.57 | 0.92 | | 0.62 | 56962 |
| weighted avg | | 1.00 | 0.99 | | 0.99 | 56962 |
| 课内_Naive Bayes 不采样_原_不调优_混淆矩阵 | | 课内_Naive Bayes 随机下采样_原_不调优_混淆矩阵 | | | 课内_Naive Bayes 过采样_原_不调优_混淆矩阵 | | |

图 27 基于拉普拉斯的贝叶斯模型混淆矩阵

不进行数据采样的情况下，贝叶斯分类器对正常交易（类别0）的分类效果较为理想，其精确率、召回率和F1分数分别达到1.00、0.99和0.99。然而，对于欺诈交易（类别1），精确率仅为0.09，但召回率达到0.81。这表明模型能够检测到大部分欺诈交易，但同时存在较多误报。

随机下采样的情况下，模型对欺诈交易的召回率进一步提升（0.91），但精确率下降至0.01，导致F1分数仅为0.02。随机下采样的策略虽然提升了对欺诈交易的敏感性，但显著增加了误报数量。

SMOTE过采样的情况下，模型对正常交易的分类效果维持在较高水平，加权平均精确率、召回率和F1分数均接近1.00。然而，对于欺诈交易（类别1），精确率从0.09提升至0.14，召回率为0.85，F1分数为0.24，宏观平均精确率、召回率和F1分数分别为0.57、0.92和0.62。尽管精确率仍然较低，但相较不采样和随机下采样，SMOTE过采样在平衡召回率与精确率之间取得了更好的折中。

### 5.2.3 随机森林（Random Forest）

使用随机森林模型对信用卡欺诈数据进行分类，并分别在不采样、随机下采样和SMOTE过采样三种情况下进行了训练和测试。以下为分类报告及混淆矩阵的结果：

表 5 随机森林分类报告

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **采样方法** | **类别** | | **Precision** | **Recall** | | **F1-Score** | **Support** |
| 不采样 | 0 | | 1.00 | 1.00 | | 1.00 | 56864 |
| 1 | | 0.91 | 0.82 | | 0.86 | 98 |
| macro avg | | 0.96 | 0.91 | | 0.93 | 56962 |
| weighted avg | | 1.00 | 1.00 | | 1.00 | 56962 |
| 随机下采样 | 0 | | 1.00 | 0.94 | | 0.97 | 56864 |
| 1 | | 0.09 | 0.87 | | 0.16 | 98 |
| macro avg | | 0.54 | 0.90 | | 0.57 | 56962 |
| weighted avg | | 1.00 | 0.94 | | 0.96 | 56962 |
| SMOTE过采样 | 0 | | 1.00 | 1.00 | | 1.00 | 56864 |
| 1 | | 0.60 | 0.83 | | 0.70 | 98 |
| macro avg | | 0.80 | 0.91 | | 0.85 | 56962 |
| weighted avg | | 1.00 | 1.00 | | 1.00 | 56962 |
| 课内_Random Forest 不采样_原_不调优_混淆矩阵 | | 课内_Random Forest 随机下采样_原_不调优_混淆矩阵  图 28 随机森林混淆矩阵 | | | 课内_Random Forest 过采样_原_不调优_混淆矩阵 | | |

不进行数据采样的情况下，随机森林模型对正常交易（类别0）的分类效果非常好，其精确率、召回率和F1分数均达到1.00。同时，模型对欺诈交易（类别1）的分类表现也较优，精确率为0.91，召回率为0.82，F1分数为0.86。这表明随机森林在处理高度不平衡数据时，对少数类的表现优于其他模型。

随机下采样的情况下，模型对欺诈交易的召回率显著提高（0.87），但精确率下降至0.09，导致F1分数仅为0.16。虽然召回率有所提高，但随机下采样导致大量误报，限制了模型的实际应用价值。

SMOTE过采样的情况下，模型对正常交易的分类效果没有不采样下好。对于欺诈交易（类别1），精确率从不采样时的0.91下降至0.60，召回率仅提升0.01。随机森林模型在不采样下表现最好。

## 5.3 课外模型

### 5.3.1 神经网络（Neural Network）

神经网络模型的具体实现基于 scikit-learn 中的 MLPClassifier。在实验中，神经网络采用了多层感知机（MLP）的结构，配置了两层隐藏层，每层神经元数量分别为 64 和 32。模型的主要参数配置如下：

表 5 神经网络参数表

|  |  |  |
| --- | --- | --- |
| 参数名称 | 值 | 参数含义 |
| hidden\_layer\_sizes | (64, 32) | 神经网络的隐藏层结构，包含两层：第一层64个神经元，第二层32个神经元。 |
| max\_iter | 1000 | 最大迭代次数，用于控制优化器训练的轮次，防止因未收敛导致训练停止。 |
| random\_state | 42 | 随机种子，用于保证实验结果的可复现性。 |
| activation | 默认值 relu | 激活函数，使用 Rectified Linear Unit (ReLU)，引入非线性特性，提高模型的表达能力。 |
| solver | 默认值 adam | 优化器，采用 Adam 方法，结合了动量梯度下降和自适应学习率，适合处理大规模数据集。 |

使用神经网络模型对信用卡欺诈数据进行分类，并分别在不采样、随机下采样和SMOTE过采样三种情况下进行了训练和测试。以下为分类报告及混淆矩阵的结果：

表 6 神经网络分类报告

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **采样方法** | **类别** | | **Precision** | **Recall** | | **F1-Score** | **Support** |
| 不采样 | 0 | | 1.00 | 1.00 | | 1.00 | 56864 |
| 1 | | 0.84 | 0.78 | | 0.80 | 98 |
| macro avg | | 0.92 | 0.89 | | 0.90 | 56962 |
| weighted avg | | 1.00 | 1.00 | | 1.00 | 56962 |
| 随机下采样 | 0 | | 1.00 | 0.95 | | 0.97 | 56864 |
| 1 | | 0.03 | 0.94 | | 0.06 | 98 |
| macro avg | | 0.52 | 0.94 | | 0.52 | 56962 |
| weighted avg | | 1.00 | 0.95 | | 0.97 | 56962 |
| SMOTE过采样 | 0 | | 1.00 | 1.00 | | 1.00 | 56864 |
| 1 | | 0.78 | 0.83 | | 0.80 | 98 |
| macro avg | | 0.89 | 0.91 | | 0.90 | 56962 |
| weighted avg | | 1.00 | 1.00 | | 1.00 | 56962 |
| Neural Network不采样_原_不调优_混淆矩阵 | | Neural Network 随机下采样_原_不调优_混淆矩阵 | | | Neural Network 过采样_原_不调优_混淆矩阵 | | |

图 29 神经网络混淆矩阵

在不进行数据采样的情况下，神经网络模型对正常交易（类别0）的分类效果非常理想，其精确率、召回率和F1分数均达到1.00。然而，对于欺诈交易（类别1），精确率为0.84，召回率为0.78，F1分数为0.80。这表明模型能够检测到大部分欺诈交易，但同时存在少量误报。

在随机下采样的情况下，模型对正常交易的召回率略有下降（0.95），但精确率仍为1.00。对于欺诈交易（类别1），召回率显著提升至0.94，但精确率大幅下降至0.03，F1分数仅为0.06。随机下采样的策略虽然显著提升了对欺诈交易的敏感性，但也导致了大量的误报。

在SMOTE过采样的情况下，模型对正常交易的分类效果仍然非常理想，精确率、召回率和F1分数均为1.00。对于欺诈交易（类别1），精确率提升至0.78，召回率为0.83，F1分数为0.80。宏观平均精确率、召回率和F1分数分别为0.89、0.91和0.90。尽管精确率仍有提升空间，但相较不采样和随机下采样，SMOTE过采样在召回率与精确率之间取得了更好的平衡，综合表现最佳。

### 5.3.2 XGBoost（eXtreme Gradient Boosting）

实验中，XGBoost的实现基于xgboost库。以下是模型的主要参数及其含义：

表 7 XGBoost参数表

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数值 | 含义 |
| eval\_metric | 'logloss' | 评估指标，使用对数损失函数进行模型评估。 |
| random\_state | 42 | 固定随机种子，保证实验结果的可复现性。 |
| max\_depth | 默认值（未调优） | 树的最大深度，用于控制模型复杂度和防止过拟合。 |
| learning\_rate | 默认值（未调优） | 学习率，用于缩小每一步梯度更新的步长。 |
| n\_estimators | 默认值（未调优） | 基学习器（树）的数量。 |

使用XGBoost内置的plot\_importance函数生成特征重要性图如下。图中，横轴表示特征的F-score值，纵轴表示特征名称。F-score越高，表示该特征在模型中的重要性越大。

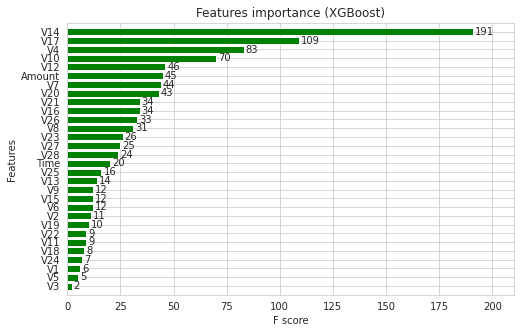


图 30 XGBoost特征重要图

从图中可以看出，某些特征的F-score显著高于其他特征，表明这些特征在区分正常交易和欺诈交易时起到了关键作用。例如，特征V191、V83、V109等具有较高的F-score值，说明它们在模型中的重要性较高，可能是欺诈检测的核心特征。相反，F-score较低的特征对模型的贡献较小，可能是冗余特征。在模型优化中可以考虑将这些得分较低的特征剔除。

使用XGBoost模型对信用卡欺诈数据进行分类，并分别在不采样、随机下采样和SMOTE过采样三种情况下进行了训练和测试。以下为分类报告及混淆矩阵的结果：

表 8 XGBoost分类报告

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **采样方法** | **类别** | | **Precision** | **Recall** | | **F1-Score** | **Support** |
| 不采样 | 0 | | 1.00 | 1.00 | | 1.00 | 56864 |
| 1 | | 0.96 | 0.78 | | 0.86 | 98 |
| macro avg | | 0.98 | 0.89 | | 0.93 | 56962 |
| weighted avg | | 1.00 | 1.00 | | 1.00 | 56962 |
| 随机下采样 | 0 | | 1.00 | 0.96 | | 0.98 | 56864 |
| 1 | | 0.04 | 0.94 | | 0.08 | 98 |
| macro avg | | 0.52 | 0.95 | | 0.53 | 56962 |
| weighted avg | | 1.00 | 0.96 | | 0.98 | 56962 |
| SMOTE过采样 | 0 | | 1.00 | 1.00 | | 1.00 | 56864 |
| 1 | | 0.80 | 0.85 | | 0.82 | 98 |
| macro avg | | 0.90 | 0.92 | | 0.91 | 56962 |
| weighted avg | | 1.00 | 1.00 | | 1.00 | 56962 |
| XGBoost不采样_原_不调优_混淆矩阵 | | XGBoost 随机下采样_原_不调优_混淆矩阵  图 31 XGBoost混淆矩阵 | | | XGBoost 过采样_原_不调优_混淆矩阵 | | |

不采样下，模型对正常交易的分类效果非常好，但对欺诈交易的召回率较低（0.78）。随机下采样下，模型显著提高了欺诈交易的召回率（0.94），但精确率大幅下降（0.04），导致F1分数较低。SMOTE过采样下，模型在召回率和精确率之间取得了较好的平衡，F1分数为0.82，综合表现最佳。

### 5.3.3 LightGBM（Light Gradient Boosting Machine）

为了提升LightGBM模型性能，我们对其重要参数进行了网格搜索，优化范围及设置如下：

表 9 LightGBM参数范围表

|  |  |  |
| --- | --- | --- |
| 参数名称 | 参数含义 | 参数值范围 |
| learning\_rate | 学习率，控制每次迭代的步长 | 0.01, 0.05, 0.1 |
| n\_estimators | 树的数量 | 100, 200, 300 |
| max\_depth | 树的最大深度 | 3, 5, 7, 10 |
| num\_leaves | 每棵树的叶子节点数 | 31, 50, 100 |
| min\_child\_samples | 一个叶子节点上数据的最小数量 | 10, 20, 30 |
| subsample | 每次迭代时训练数据的随机采样比例 | 0.6, 0.8, 1.0 |
| colsample\_bytree | 每棵树随机选择的特征占比 | 0.6, 0.8, 1.0 |
| reg\_alpha | L1正则化项 | 0, 0.1, 0.5 |
| reg\_lambda | L2正则化项 | 0, 0.1, 0.5 |

通过网格搜索，最终选择了learning\_rate=0.05，n\_estimators=200，max\_depth=7，num\_leaves=50等参数的组合，效果最佳。

使用Gain Importance作为特征重要性度量指标，并调用LightGBM的plot\_importance()函数生成特征重要性图如下。

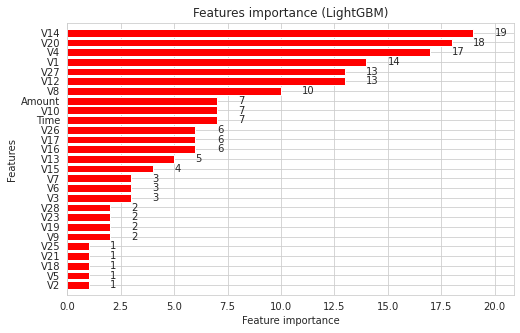


图 32 LightGBM特征重要图

由LightGBM特征重要图可以看出，特征重要性排序与XGBoost模型得到的重要性排序有很大的不同，但有些特征如V14、V4等始终排在前面，说明这两个特征可能与类别的关系更加紧密。而有些特征在两张图中的位置相差甚远，如V17在XGBoost特征重要图中排名第二，LightGBM特征重要图排行缺并不靠前，说明特征的重要性还需要进一步定夺，综合考虑不同模型的结果。

表 10 LightGBM分类报告

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **采样方法** | **类别** | **Precision** | **Recall** | **F1-Score** | **Support** |
| 不采样 | 0 | 1.00 | 1.00 | 1.00 | 56864 |
| 1 | 0.23 | 0.60 | 0.33 | 98 |
| macro avg | 0.61 | 0.80 | 0.66 | 56962 |
| weighted avg | 1.00 | 1.00 | 1.00 | 56962 |
| 随机下采样 | 0 | 1.00 | 0.97 | 0.98 | 56864 |
| 1 | 0.04 | 0.94 | 0.09 | 98 |
| macro avg | 0.52 | 0.95 | 0.53 | 56962 |
| weighted avg | 1.00 | 0.97 | 0.98 | 56962 |
| SMOTE过采样 | 0 | 1.00 | 1.00 | 1.00 | 56864 |
| 1 | 0.56 | 0.85 | 0.68 | 98 |
| macro avg | 0.78 | 0.92 | 0.84 | 56962 |
| weighted avg | 1.00 | 1.00 | 1.00 | 56962 |

|  |  |  |
| --- | --- | --- |
| LightGBM不采样_原_不调优_混淆矩阵 | LightGBM 随机下采样_原_不调优_混淆矩阵 | LightGBM 过采样_原_不调优_混淆矩阵 |

图 33 LightGBM混淆矩阵

不采样下，模型对正常交易的分类效果非常好，但对欺诈交易的召回率较低（0.60），且精确率较低（0.23）。随机下采样下，模型显著提高了欺诈交易的召回率（0.94），但精确率极低（0.04），导致F1分数非常低。SMOTE过采样下，模型在召回率和精确率之间取得了较好的平衡，F1分数为0.68，综合表现优于不采样和随机下采样。

### 5.3.4 AutoEncoder + 逻辑回归

对数据进行预处理后，为了平衡数据集，从样本中随机选择1000个样本（由于欺诈样本较少，保留所有欺诈样本）。然后，我们将非欺诈样本与欺诈样本合并，并打乱数据。利用T-SNE进行数据降维并可视化，得到如下图：

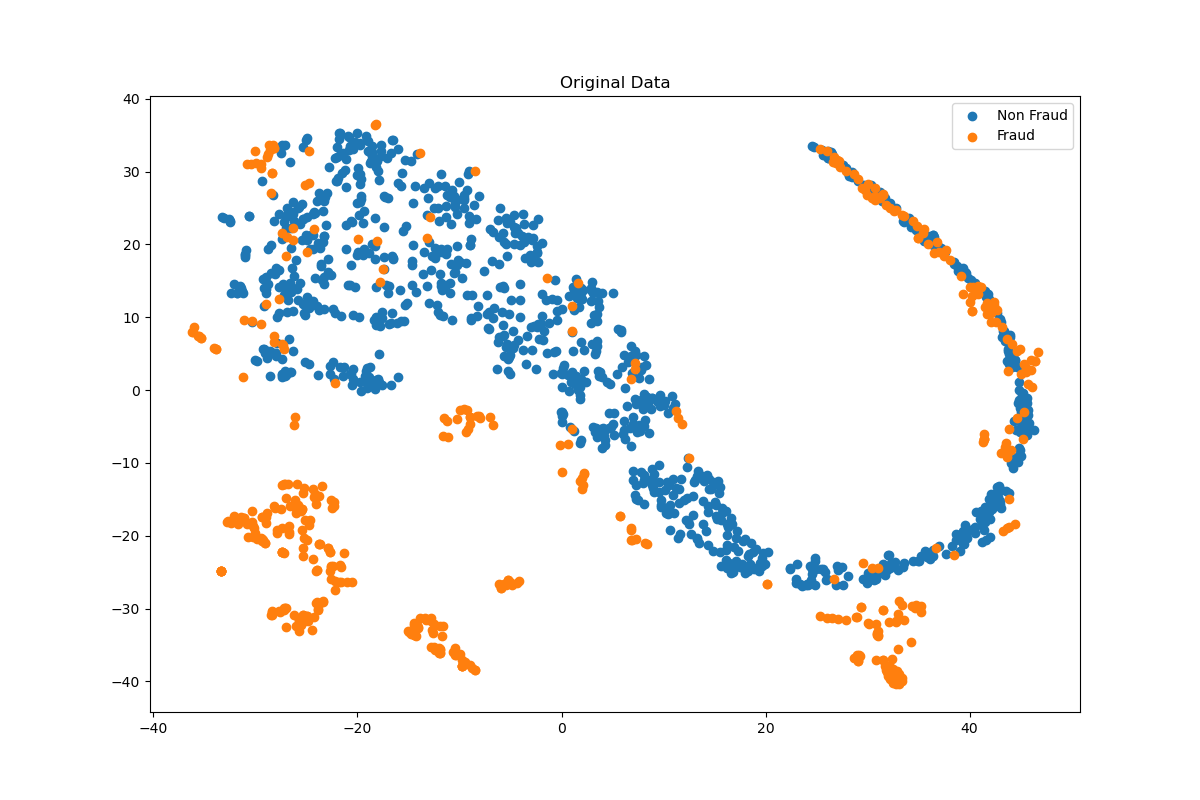


图 34 AutoEncoder训练前数据

在图25中，可以看到通过t-SNE降维后的数据点分布，其中绿色表示非欺诈样本，红色表示欺诈样本。从图中可以看出，数据中的两类样本在二维空间中的分布较为分散，且存在一定的重叠，这表明模型需要提取有效的特征来区分这两类样本。

使用AutoEncoder对数据进行降维和特征提取。AutoEncoder通过学习数据的低维表示，将原始数据压缩成较小的维度，同时保留数据的关键特征。经过训练后的 AutoEncoder 输出的是一个包含低维表示的数据集。

在本实验中，AutoEncoder模型的网络结构由编码器和解码器两部分组成。输入层的维度与原始数据的特征数相同。编码器包含两层，第一层将输入数据映射到100个神经元，并使用Tanh激活函数进行非线性转换；第二层进一步将数据降维到50个神经元，并使用ReLU激活函数。解码器部分则反向操作，第一层将50维的数据映射回100维，使用Tanh激活函数；第二层将100维的数据还原到原始输入的维度，使用ReLU激活函数。模型的优化器采用Adam优化器，学习率设为0.001，损失函数选择均方误差（MSE），用于衡量输入数据与重构数据之间的差异，优化模型的训练效果。对模型训练后的数据再次进行可视化，得到下图：

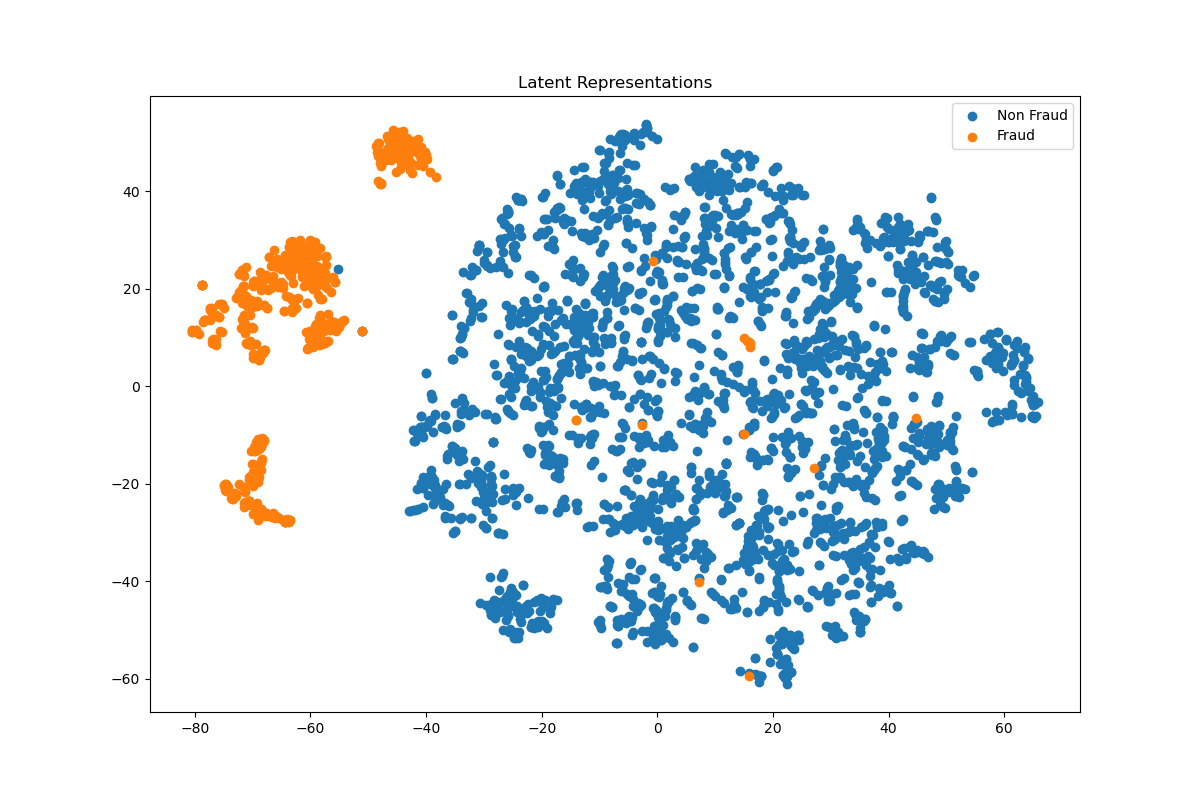


图 35 AutoEncoder训练后数据

在图26中，可以看到经过AutoEncoder 训练后的数据点分布。与图25相比，经过特征提取后，数据的两类样本变得更加可区分，红色（欺诈样本）和绿色（非欺诈样本）在低维空间中的分布变得更加分离。

最后，利用逻辑回归模型，对数据进一步判别，得到分类报告、混淆矩阵、ROC曲线如下。

Classification Report:

precision recall f1-score support

0.0 0.98 1.00 0.99 754

1.0 1.00 0.87 0.93 119

micro avg 0.98 0.98 0.98 873

macro avg 0.99 0.94 0.96 873

weighted avg 0.98 0.98 0.98 873

Accuracy Score: 0.9828178694158075

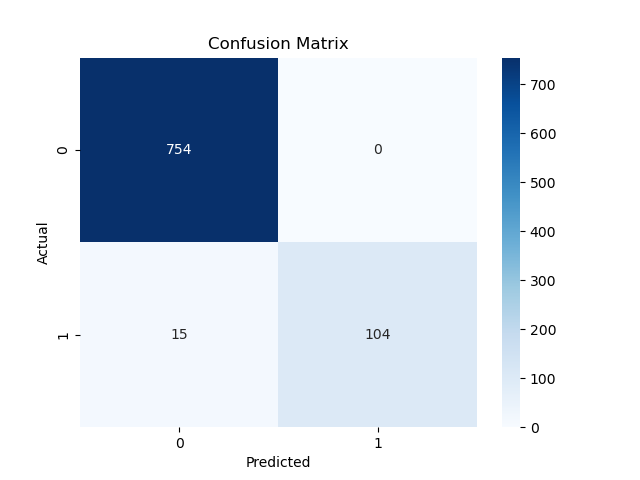
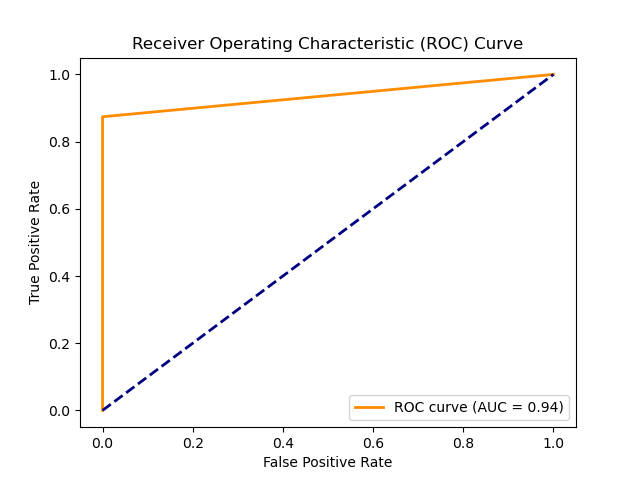


图 36 混淆矩阵 图 37 ROC曲线

从分类报告和准确率来看，模型在信用卡欺诈检测任务中表现优异。对于正常交易（类别0），模型的精确率和召回率均接近100%，表明模型能够准确识别正常交易。对于欺诈交易（类别1），虽然精确率为100%，但召回率为87%，说明模型能够有效识别大部分欺诈交易，但仍存在少量漏检。总体准确率为98.28%，表明模型在整体分类任务中表现良好。然而，由于欺诈交易的样本较少，召回率的提升仍然是进一步优化的重点，以减少欺诈交易的漏检率。

# 六、实验结果分析

## 6.1模型评估

在本节中，我们对不同模型在三种数据预处理方法（未采样、随机下采样、SMOTE过采样）下的表现进行汇总分析，重点对比准确率 (Accuracy)、精确度 (Precision)、召回率 (Recall) 和 F1 分数 (F1-Score) 等指标，考虑到数据集严重不平衡的问题，我们采用Macro Avg值。同时，通过绘制各模型的ROC曲线和结果直方图，直观展示模型性能。

### 6.1.1 不采样数据

表 11 不采样数据模型结果表

| 模型 | Accuracy | Precision | Recall | F1-Score |
| --- | --- | --- | --- | --- |
| KNN | 1.00 | 0.97 | 0.89 | 0.92 |
| Bayes | 0.98 | 0.53 | 0.90 | 0.55 |
| Random Forest | 1.00 | 0.99 | 0.88 | 0.93 |
| Neural Network | 1.00 | 0.92 | 0.89 | 0.90 |
| XGBoost | 1.00 | 0.98 | 0.89 | 0.93 |
| LightGBM | 1.00 | 0.61 | 0.80 | 0.66 |

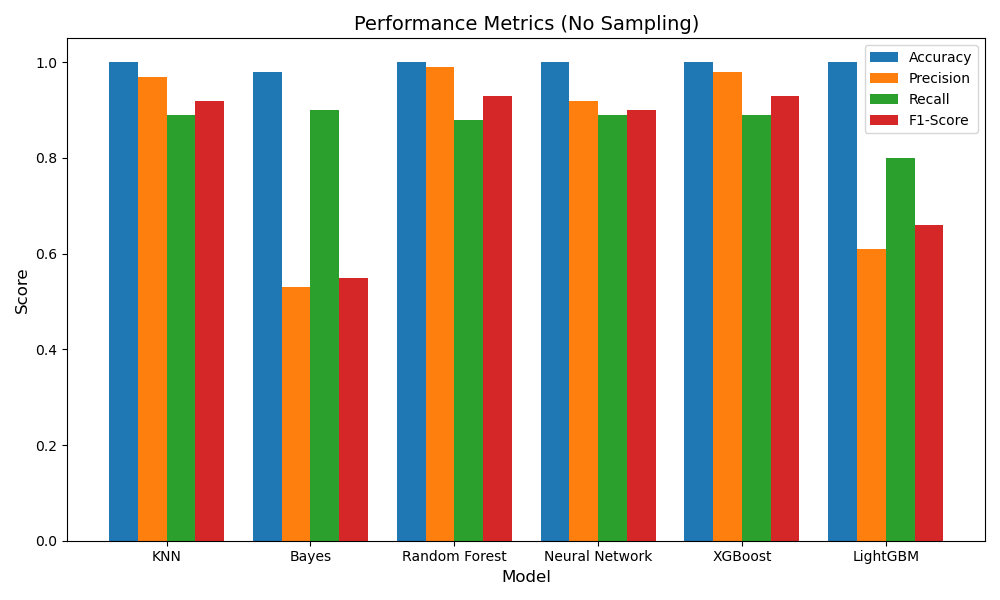


图 38 不采样数据模型结果直方图

未进行采样处理的数据中，所有模型的准确率表现普遍较高，尤其是KNN、Random Forest、Neural Network、XGBoost和LightGBM的准确率都达到了1.00。但从精确率(Precision)和F1分数(F1-Score)上看，模型间的差异较大，其Bayes和LightGBM的精确率较低，而Random Forest和XGBoost在各项指标中表现最佳。且所有模型的召回率都比较低，可见还有很大的提升空间。

### 6.1.2 随机下采样数据

表 12 随机下采样数据模型结果表

| 模型 | Accuracy | Precision | Recall | F1-Score |
| --- | --- | --- | --- | --- |
| KNN | 0.98 | 0.53 | 0.94 | 0.55 |
| Bayes | 0.97 | 0.53 | 0.91 | 0.54 |
| Random Forest | 0.98 | 0.53 | 0.96 | 0.55 |
| Neural Network | 0.95 | 0.52 | 0.94 | 0.52 |
| XGBoost | 0.96 | 0.52 | 0.95 | 0.53 |
| LightGBM | 0.97 | 0.52 | 0.95 | 0.53 |

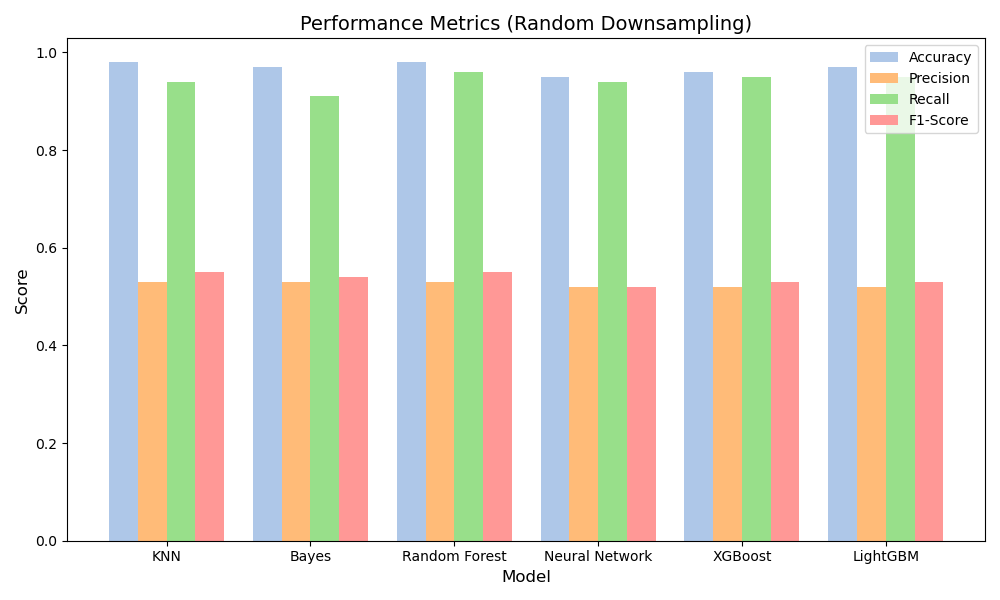


图 39 随机下采样数据模型结果直方图

在随机下采样的数据中，由于样本数量减少，所有模型的准确率相比未采样数据有所下降，但整体上大大增加了所有模型的召回率。其中，**KNN和Random Forest**的召回率较高（均为**0.94**或以上），但**精确率**和**F1分数**出现明显下降，表明随机下采样可能导致正负样本的平衡性降低，从而影响模型性能的全面性。

### 6.1.3 SMOTE过采样数据

表 13 SMOTE过采样数据模型结果表

| 模型 | Accuracy | Precision | Recall | F1-Score |
| --- | --- | --- | --- | --- |
| KNN | 1.00 | 0.74 | 0.93 | 0.81 |
| Bayes | 0.98 | 0.53 | 0.92 | 0.55 |
| Random Forest | 1.00 | 0.96 | 0.92 | 0.94 |
| Neural Network | 1.00 | 0.89 | 0.91 | 0.90 |
| XGBoost | 1.00 | 0.90 | 0.92 | 0.91 |
| LightGBM | 1.00 | 0.78 | 0.92 | 0.84 |

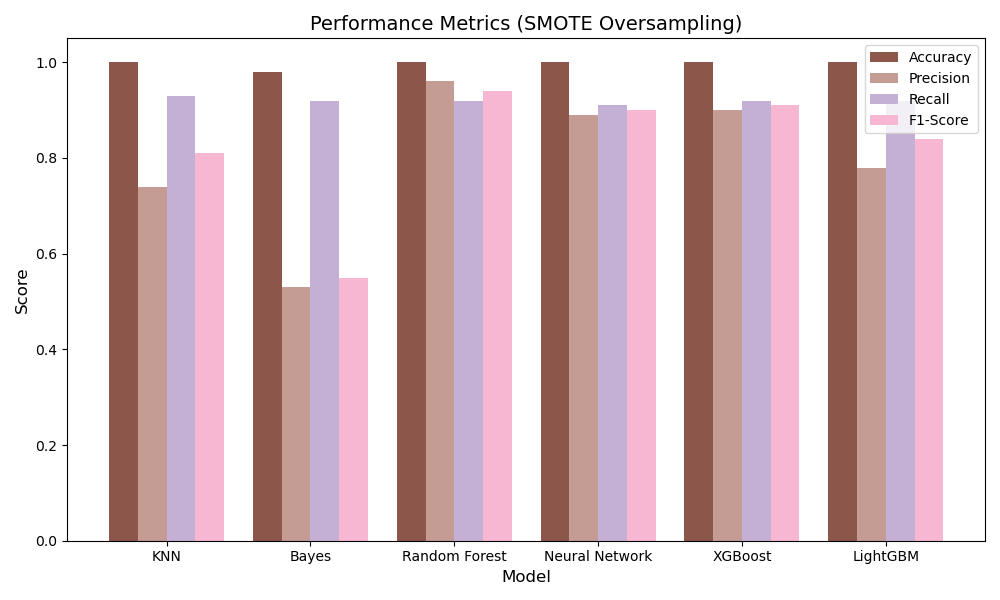


图 40 SMOTE过采样数据模型结果直方图

在使用SMOTE过采样处理后的数据中，模型表现显著改善，特别是KNN 和LightGBM的精确率和F1分数有了明显提升。此外，Random Forest和XGBoost在所有指标中均表现出色，尤其是F1分数达到了0.94和0.91，表明SMOTE 过采样能有效缓解数据不平衡问题，提升模型对少数类别的预测能力。

|  |  |
| --- | --- |
| 课外3种模型ROC曲线汇总 | 课内模型ROC曲线汇总 |

图 41 模型ROC曲线汇总

为更直观地评估各模型的性能，我们绘制了不同数据预处理方法下的**ROC曲线**。从图中可以看出，所有模型在未采样和SMOTE过采样数据上的 AUC 值均较高，表明模型在区分正负样本方面具有较强能力；而随机下采样数据上的 AUC 值稍有下降。

## 6.2采样方法影响

|  |  |  |
| --- | --- | --- |
| KNN不同采样方式直方图 | Bayes不同采样方式直方图 | Random Forest不同采样方式直方图 |
| Neural Network不同采样方式直方图 | XGBoost不同采样方式直方图 | LightGBM不同采样方式直方图 |

图 42 不同模型的结果直方图

在原始数据集上，KNN和神经网络等模型的表现优于采样后的结果。同时，XGBoost在原始数据集上的表现也优于经过两种采样处理后的效果，这可能是因为XGBoost本身对不平衡数据集具有较强的适应能力。

对于大多数模型而言，随机下采样的效果并不理想，尽管提高了Recall值，但其他指标普遍有所下降。这种情况通常是因为随机下采样通过减少多数类样本数量来平衡数据集，从而损失了部分多数类样本的信息，导致模型在整体分类能力上有所下降。

而经过SMOTE过采样后，许多模型的F1分数与AUC值都有大幅提升，如LightGBM模型的Recall值从原本的0.80提升到了0.92，大大提高了模型性能。SMOTE通过生成新的少数类样本，显著改善了模型对少数类的识别能力，尤其是在F1分数和AUC值上的表现。

总体来看，除了结合AutoEncoder与逻辑回归的方法外，随机森林和XGBoost模型在处理不平衡数据集时表现最为优秀，且在原始数据集、随机下采样和SMOTE过采样后的表现都较为稳定。这表明，这两种模型具有较强的泛化能力和对不平衡数据的适应性。特别是XGBoost，通过梯度提升框架和灵活的参数调整，能够有效缓解类别不平衡问题。随机森林则通过多样化的决策树集成方法，保持了较高的准确率和稳定性。

# 七、解决问题及主要收获

## 7.1问题解决

在本次实验中，我遇到了一系列关键问题，但都通过一系列有效的技术手段解决了，且取得了显著的成果。

首先，针对数据不平衡问题，我尝试了多种采样方法，包括随机下采样和SMOTE过采样。这些方法在平衡数据分布的同时，尽可能避免了因样本不足或过拟合带来的问题，从而有效提升了模型对少数类样本的分类性能。

其次，为了减少特征冗余并提高模型训练效率，我通过特征降维技术优化了数据集。具体来说，使用主成分分析（PCA）和AutoEncoder等方法。PCA通过线性变换提取了数据中最具代表性的主成分，显著减少了数据维度；而AutoEncoder则利用深度学习模型，从数据中学习到了潜在的非线性特征。这两种方法的结合不仅减少了计算复杂度，还提升了模型对关键特征的理解能力。

最后，AutoEncoder的应用在本实验中取得了优秀的成绩，综合评分是最高的。通过构建多层神经网络，AutoEncoder能够自动学习数据中的深层特征结构，从而进一步提升了模型的检测准确性。特别是在处理高维、复杂数据时，AutoEncoder展现了其独特的优势，为实验结果的提升提供了重要支持。

## 7.2主要收获

在本次实验中，我围绕一个具有挑战性的课题——处理严重数据不平衡的大规模数据集，进行了深入的探索和实践。从数据预处理到模型训练，再到评估与优化，我不仅解决了具体的问题，还收获了许多宝贵的经验和技能。

通过本次实验，我第一次接触了严重数据不平衡的大规模数据集，同时也学习到了许多对于不平衡数据集的处理方法，包括数据采样、数据异常值处理、数据特征相关性分析等等。我了解并实践了随机下采样和SMOTE过采样等技术。其中，随机下采样通过减少多数类样本数量来平衡数据分布，但可能会丢失部分数据的信息；而SMOTE通过生成少数类的合成样本，显著提升了模型对少数类的分类能力，同时保留了数据的完整性。这些技术让我更直观地理解了如何在样本分布极不平衡的情况下增强模型的泛化能力。在学习这些数据处理方法时，我查询了许多文献与资料，而有关该数据集的介绍大多是英文撰写的，在这一过程中也极大地提高了我的英文阅读水平。

同时，本次实验我运用了许多机器学习模型，不仅仅是课内学习过的模型，我还另外学习到了许多课内所没有接触到的模型。除了常见的KNN、随机森林、XGBoost等算法外，我还尝试了LightGBM等模型。这让我更加直观地感受到不同模型在实际应用中的表现差异。例如，在处理不平衡数据时，树模型（如XGBoost和LightGBM）相较于简单的线性模型更能有效应对数据复杂性和类别不平衡问题。将机器学习模型其运用到实际项目中，这不仅提高了我对课内模型的理解，同时也拓宽了我的知识面，了解与熟悉了许多课内所没有接触过的模型，如LightGBM等模型。

由于数据极度不平衡，我还掌握了模型评估的其他方法。不同于之前所关注的准确率，本数据集更加关注少数类精准率以及召回率，为此我还查阅了许多关于数据不平衡的参考文献，从中学习到了很多关于数据不平衡处理的方法。

# 八、参考来源

1. [Credit Card Fraud Detection](https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud)
2. [Credit Fraud || Dealing with Imbalanced Datasets](https://www.kaggle.com/code/janiobachmann/credit-fraud-dealing-with-imbalanced-datasets)
3. [FraudDetection 99.9% accuracy](https://www.kaggle.com/code/vishwasmishra1234/frauddetection-99-9-accuracy)
4. [Credit Card Fraud Detection Predictive Models](https://www.kaggle.com/code/gpreda/credit-card-fraud-detection-predictive-models)
5. 周志华. 机器学习[M]. 清华大学出版社, 2016.
6. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T. Y. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. NIPS, 2017.
7. [【机器学习】XGBoost数学原理及详细实现过程\_xgboost的数学原理-CSDN博客](https://blog.csdn.net/qq_70699891/article/details/134257083)
8. Ke, Guolin, et al. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree." Advances in Neural Information Processing Systems (NeurIPS), 2017.
9. Haibo He,Member.Learning from Imbalanced Data[J].IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING(TKDE),2009,21(9):1263-1284
10. S. Dhankhad, E. Mohammed and B. Far, "Supervised Machine Learning Algorithms for Credit Card Fraudulent Transaction Detection: A Comparative Study," 2018, pp. 122-125, doi: 10.1109/IRI.2018.00025.
11. M. Zareapoor and P. Shamsolmoali, “Application of credit card fraud detection: Based on bagging ensemble classifier,” Procedia Computer Science, vol. 48, pp. 679–685, 2015
12. Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the Dimensionality of Data with Neural Networks. Science.
13. O. S. Yee, S. Sagadevan, and N. H. A. H. Malim, “Credit card fraud detection using machine learning as data mining technique,” Journal of Telecommunication, Electronic and Computer Engineering (JTEC), vol. 10, no. 1-4, pp. 23–27, 2018.
14. Principal Component Analysis, Wikipedia Page, <https://en.wikipedia.org/wiki/Principal_component_analysis>
15. RandomForrestClassifier, <http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>
16. ROC-AUC characteristic, <https://en.wikipedia.org/wiki/Receiver_operating_characteristic#Area_under_the_curve>
17. XGBoost Python API Reference, <http://xgboost.readthedocs.io/en/latest/python/python_api.html>
18. LightGBM Python implementation, <https://github.com/Microsoft/LightGBM/tree/master/python-package>
19. LightGBM algorithm, <https://www.microsoft.com/en-us/research/wp-content/uploads/2017/11/lightgbm.pdf>
20. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.