

華東師範大學
East China Normal University

本科生毕业论文

基于不平衡数据的信用卡欺诈检测研究 Research on Bank Credit Card Default Prediction on a Imbalanced Data

姓 名: 夏嘉琦

学 号: 10174602225

学 院: 统计学院

专 业: 统计学

指导老师: 孙 蕾

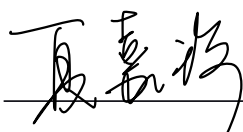
职 称: 副 教 授

完成时间: 2022 年 4 月

华东师范大学学位论文诚信承诺

本毕业论文是本人在导师指导下独立完成的，内容真实、可靠。本人在撰写毕业论文过程中不存在请人代写、抄袭或者剽窃他人作品、伪造或者篡改数据以及其他学位论文作假行为。

本人清楚知道学位论文作假行为将会导致行为人受到不授予/撤销学位、开除学籍等处理（处分）决定。本人如果被查证在撰写本毕业论文过程中存在学位论文作假行为，愿意接受学校依法作出的处理（处分）决定。

承诺人签名： 

日期： 2022 年 5 月 1 日

华东师范大学学位论文使用授权说明

本论文的研究成果归华东师范大学所有，本论文的研究内容不得以其它单位的名义发表。本学位论文作者和指导教师完全了解华东师范大学有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅；本人授权华东师范大学可以将论文的全部或部分内容编入有关数据库进行检索、交流，可以采用影印、缩印或其他复制手段保存论文和汇编本学位论文。

保密的毕业论文（设计）在解密后应遵守此规定。

作者签名：  导师签名：_____ 日期： 2022 年 5 月 1 日

目录

摘要.....	1
Abstract.....	2
1. 引言与文献综述.....	3
1.1 研究背景及意义.....	3
1.2 国内外研究现状.....	3
1.3 论文研究内容.....	5
1.4 论文基本结构.....	5
2. 研究方法及相关理论基础.....	6
2.1 不平衡数据处理方法.....	6
2.1.1 随机过采样与随机欠采样.....	6
2.1.2 SMOTE 过采样.....	6
2.1.3 自适应合成采样(ADASYN).....	6
2.2 机器学习模型.....	7
2.2.1 逻辑回归.....	7
2.2.2 随机森林.....	8
2.3 不平衡问题的分类评价标准.....	8
2.3.1 召回率(recall)、精确度(precision)及 F1 分数.....	8
2.3.2 ROC 曲线和 AUC 面积.....	9
3. 数据分析.....	11
3.1 数据集介绍.....	11
3.2 探索性数据分析.....	11
3.2.1 各变量的相关关系.....	11
3.2.2 数据归一化处理.....	12
3.2.3 特征选取.....	14
4. 建模分析.....	15
4.1 逻辑回归.....	15
4.2 随机森林.....	15
4.3 结果分析.....	16
5. 结论与展望.....	18
参考文献.....	19
附录.....	21
致谢.....	36

基于不平衡数据的信用卡欺诈检测研究

摘要

随着现代科技的发展和电子商业的不断壮大，信用卡欺诈问题也日益严重。随着大数据和人工智能的发展，机器学习方法成为监测信用卡欺诈的主流方法。但与此同时，挑战现有技术和系统的新型诈骗方法也层出不穷，消耗了大量人力财力。因此，构建一个能够快速准确分辨少数信用卡违约交易的模型对银行和客户都至关重要。在本研究中，使用了经过 PCA 转换的欧洲信用卡数据集。这类研究中经常会出现不平衡数据集，即数据集中，正常交易数量远多于违约交易数量，如果采用传统的机器学习模型会导致结果偏向多数类数据，从而结果准确性出现偏差。本文采用了随机过采样、随机欠采样、SMOTE 过采样以及 ADASYN 自适应合成采样几种方法处理不平衡数据，然后使用逻辑回归和随机森林方法进行模型训练。通过对比，基于随机过采样的随机森林方法表现最好，在所有组合中 F_1 分数最高，为 0.8679。

关键词：信用卡欺诈、不平衡数据、机器学习

Research on Bank Credit Card Default Prediction on a Imbalanced Data

Abstract

Credit card fraud is becoming increasingly serious with the development of modern technology and the global superhighways of e-commerce. With the development of data mining and machine learning technology, machine learning methods have become a mainstream method to detect credit card fraud. Credit card fraudsters continuously try to come out with a new tactic challenged the present technology and system, which costs a lot of human resources and money to deal with. Therefore, a quick and accurate model that can distinguish a small number of credit card default transactions become essential for credit card providers and customers. In this study, a real-world dataset of European credit card transactions, with PCA transformation applied is being used. One common problem happened in this kind of research is the data tend to be imbalanced, that is, in the data set, the number of normal transactions is much more than the number of default transactions. If we apply traditional machine learning model without any modification, imbalanced data will often introduce bias which the accuracy of the prediction is not accurate. In this study, we used random oversampling, random undersampling, SMOTE oversampling and ADASYN adaptive synthetic sampling to deal with imbalanced data. And the model has been trained with Logistic Regression and Random Forest. In conclusion, Random Forest with Oversampling technique is our final model, as this gives highest F_1 score of 0.8679 on test datasets.

Key Words: Credit Card, Imbalanced Data, Machine Learning

1. 引言与文献综述

1.1 研究背景及意义

1887 年, 爱德华·贝拉米(Edward Bellamy)在他的小说《向后看》中首次描述了使用信用卡购物的概念。1930 年以来, 美国开始出现早期信用卡的雏形, 如航空旅行卡、通用收费卡等, 用于分期付款和支付账单。1958 年, 美国银行在加利福尼亚州推出了 BankAmericard, 是世界上第一张可识别的现代信用卡。20 世纪后期, 信用卡在美国、英国、加拿大达到了非常高的使用率, 信用卡的无现金支付形式开始流行起来。信用卡因其简单方便的操作、物理上的便携性以及信用卡号的隐私性和安全性, 逐渐在全世界风靡起来^[1]。

随着信用卡在全球的交易大幅增加和电子商业的不断发展, 信用卡欺诈问题也日益严重。信用卡欺诈的形式主要分为三种: 一、未经持卡人的同意使用他人信用卡交易; 二、利用他人资料申请信用卡; 三、伪造信用卡实施诈骗。因此, 对信用卡欺诈行为的监测和识别就尤为重要, 不仅能减少持卡人和发卡机构的经济损失, 而且能降低金融机构的经济风险, 有利于国家风险管理和金融行业长足发展^[2]。

近年来, 随着大数据和人工智能的发展, 机器学习方法成为监测信用卡欺诈的主流方法。通过数据挖掘技术, 根据持卡人的历史交易记录和交易金额, 构建能有效分辨正常交易和可疑交易的模型。在监测信用卡欺诈问题中, 经常会出现不平衡数据集, 即正常交易数量远多于违约交易数量。如不对数据进行处理, 传统的机器学习模型会偏向于多数类数据以确保模型的有效性。此时, 构建一个能够分辨极少数违约交易的模型, 减少错误分类带来的损失, 就对处理信用卡欺诈问题尤为重要。

1.2 国内外研究现状

信用卡欺诈检测方法分为两大类: 有监督方法和无监督方法^[3]。在有监督的欺诈检测方法中, 根据欺诈交易和正常交易的训练样本建立模型, 来预测新交易的分类。在无监督的欺诈检测方法中, 异常值或异常交易被识别为潜在的欺诈交易。

近年来, 随着科学技术的大量应用, 研究人员提出了各种基于人工智能和机器学习来检测信用卡欺诈的方法: Ghosh 和 Reilly (1994) ^[4]使用基于神经网络的信用卡欺

诈检测系统对大量信用卡交易样本进行训练,他们发现,相比于基于规则的检测信用卡欺诈程序,这种基于神经网络的模型检测到的信用卡欺诈交易显著增加,误报率显著减少。Srivastava 等人(2012)^[5]使用隐马尔可夫模型(HMM)对信用卡交易中的交易特征序列进行建模,证明了其在欺诈检测方面的有效性。Sahin 和 Duman (2008)^[6]就检测信用卡欺诈问题对决策树和支持向量机(SVM)进行了比较。他们把一个数据集分为三组,其中欺诈交易和正常交易的比例不同,他们一共开发了七个基于决策树和 SVM 的模型并使用以上数据集进行测试。实验结果表明,基于决策树的模型优于基于 SVM 的模型。但是,随着训练集的样本数量增加,基于 SVM 的模型可达到与基于决策树的模型差不多的准确性。Mishra 和 Ghorpade(2018)^[7]评估了各种基于各种分类和集成技术进行训练的模型,比如:逻辑回归、决策树、随机森林、支持向量机 SVM 和各种集成模型。实验结果表明,随机森林算法,召回率(recall)约为 96%。Xuan 等人(2018)^[8]使用基于随机树的随机森林和基于 CART 的随机森林模型来区分正常交易和可疑交易。他们使用这两种模型对不同数据集进行了三次实验。结果表明,基于 CART 的随机森林模型表现更好。

国内的信用卡欺诈检测起步较晚,吴婷(2006)^[9]使用基于支持向量机和决策树的组合分类器识别信用卡欺诈,用 K-means 聚类方法解决样本不对称分布问题,并使用 Adacost 进行分类结果融合。杨玺(2008)^[10]构建了一个基于支持向量机 SVM 的欺诈交易检测模型来检测高风险信用卡交易行为,检测信用卡欺诈问题。刘艳红(2010)^[11]提出一种基于单值分类方法的支持向量数据描述算法(SVDD),并把 K-means 聚类方法与改进的 SVDD 算法结合生成一种基于 KmD-SVDD 算法的两阶段信用卡欺诈检测模型。陈冠宇等(2018)^{[12] [13]}结合了长短期记忆网络、SMOTE 过采样和 KNN 分类算法构建了一个信用卡欺诈检测模型,可以通过 KNN 筛选出安全生成样本来提升模型性能,克服了 SMOTE 过采样在生成新样本时的局限性。黄勇新(2020)^[14]把深度森林算法和一种新的综合采样算法结合构建了一个信用卡欺诈检测模型,该算法利用高斯混合分布和 K 近邻的思想对多数类数据进行下采样,再根据自适应合成采样 ADASYN 对少数类数据向上采样,最后综合两部分数据形成新的训练集。

从上述回顾可以看到,使用不同的机器学习方法会产生很好的效果,但是如果可以用更少的资源实现相同甚至更好的结果呢?也许通过适当的预处理不同机器学习方法也可以给出不错的结果。由于信用卡数据集是一个极不平衡数据集,所以首先可以使用抽样方法对数据集进行平衡化处理。本文决定比较随机过采样、随机欠采样、

SMOTE 过采样和自适应合成采样(ADASYN)四种方法以及逻辑回归和随机森林模型对信用卡欺诈检测的适用性。

1.3 论文研究内容

本文把处理不平衡数据的方法和机器学习算法结合, 对比了几种方法在处理信用卡欺诈检测问题上的优劣。本文的研究过程分为四个阶段:

1. 数据预处理。V1, V2,... V28 这 28 个特征经过 PCA 变换, 特征之间相互独立。把 Amount 列进行归一化, 减小数值浮动差异, 保证特征之间分布差异相似, 重要程度相当。
2. 数据平衡化处理。在划分好训练集和测试集后, 由于本文数据为极度不平衡样本集, 对于不平衡样本, 采用随机过采样、随机欠采样、SMOTE 过采样和自适应合成采样(ADASYN)方法进行数据平衡化处理。
3. 基于机器学习方法建立信用卡欺诈模型。本文使用的算法有: 逻辑回归和随机森林算法。汇总各个模型准确度 (accuracy)、召回率 (recall)、精确度 (precision)、混淆矩阵 (confusion matrix) 等, 并绘制 ROC 曲线。
4. 模型结果分析。对比各个模型的 F_1 分数和运行时间, 确定最佳的信用卡欺诈监测模型。

1.4 论文基本结构

1. 引言与文献综述。主要介绍了研究背景及意义、国内外研究现状、论文的研究内容及基本结构。
2. 研究方法及相关理论基础。介绍了与本文相关的数据层面上的不平衡数据处理方法; 概述了机器学习中逻辑回归和随机森林基本原理。
3. 数据分析。介绍了本文使用的 Kaggle 信用卡数据集的基本信息, 并对数据进行预处理和特征选取, 划分训练集和测试集后, 对数据做了平衡化处理。
4. 建模分析。把机器学习方法和数据平衡化处理方法相结合进行建模, 通过对比各个模型的 F_1 分数和运行时间, 确定最佳的信用卡欺诈监测模型。
5. 总结与展望。主要指出来本文的局限与不足。

2. 研究方法及相关理论基础

2.1 不平衡数据处理方法

2.1.1 随机过采样与随机欠采样

在数据层面处理不平衡数据最简单有效的方法是随机欠采样和随机过采样。随机过采样从少数类样本中随机选择样本，替换后添加到训练集中。随机欠采样从多数类样本中随机选择样本，并将它们从训练集中删除。这两种方法对数据没有任何假设，易于实现而且执行速度快，适用于庞大复杂的数据集。但这两种方法也有一些缺点，随机过采样只是简单地增加了样本数量，并没有为数据集增加任何新信息；随机欠采样可能删除数据集中一些重要的信息，增加分类器的方差，影响结果的准确率。

2.1.2 SMOTE 过采样

SMOTE 过采样算法是由 Chawla 等(2002)^[15] 提出的，基于 KNN 算法，由在少数类样本之间进行插值来产生额外的样本。具体地，对于每个少数类样本，计算出 K 个近邻 (k 值需要提前指定，一般设置成 5)，从 K 个近邻中挑选 N 个样本进行随机线性插值，生成新的合成样本。其算法过程如下^[16]：

1. 设少数类样本的集合为 A，对其中的每一个少数类样本 x_i ，计算 x_i 与 A 中所有其他样本之间的欧氏距离，得到 x_i 的 k 近邻。
2. 根据样本不平衡比例设置采样倍率 N，对每个 A 中样本 x_i ，从其 K 近邻中随机选择 N 个样本(x_1, x_2, \dots, x_n)，记为集合 A_1 。
3. 对于每个随机选择的近邻 $x_k \in A_1$ ，分别与原样本 x_i 按照如下的公式构建新样本 x_{new} (其中 $\text{rand}(0,1)$ 表示 0 到 1 之间的随机数)：

$$x_{new} = x_i + \text{rand}(0,1) * |x_k - x_i| \quad (1)$$

2.1.3 自适应合成采样 (ADASYN)

ADASYN 自适应合成采样算法由 He 等(2008)^[17] 提出，和 SMOTE 过采样算法类似，也是由少数类样本产生合成样本。算法步骤如下^[16]：

1. 计算不平衡度，即少数类与多数类的比率：记 m_s 为少数类样本的数量， m_l 为

少数类样本的数量，则不平衡度为：

$$d = m_s/m_l, \text{ where } d \in [0, 1] \quad (2)$$

2. 如果 $d < d_{th}$ ，其中 d_{th} 是最大容许不平衡率的预设阈值：

(1) 计算要合成的少数类样本总数：

$$G = \beta(m_l - m_s), \beta \in [0, 1] \quad (3)$$

β 是 ADASYN 合成数据后所需的不平衡度。当 $\beta=1$ 时，即 G 等于少数类和多数类的差值，表示 ADASYN 后，数据集完美平衡。

(2) 对于每个少数类样本 x_i ，找到其 K 近邻，并计算比例 r_i ：

$$r_i = \Delta_i/K, i=1,2,\dots,m_s \quad (4)$$

其中 Δ_i 是 x_i 的 K 近邻中的多数类样本的数量。 r_i 值越大表示 x_i 的 K 近邻包含的多数类样本更多，并且更难学习。

(3) 标准化 r_i ，使其总和为 1：

$$\hat{r}_i = r_i / \sum_{i=1}^{m_s} r_i \quad (5)$$

(4) 计算每个少数类样本 x_i 需要生成的合成样本的数量：

$$g_i = \hat{r}_i \times G \quad (6)$$

由于 r_i 值越大表示 x_i 的 K 近邻包含的多数类样本更多，因此这些邻域将生成更多的合成少数类示例。这就是为什么 ADASYN 算法具有自适应性：为“更难学习”的 x_i 的 K 近邻生成更多合成样本。

(5) 在每个待合成的少数类样本 x_i 的 K 近邻中选择另一个少数类样本 x_{zi} ，可以使用以下公式计算新的合成样本：

$$s_i = x_i + \lambda(x_{zi} - x_i) \quad (6)$$

其中， λ 是 0 到 1 之间的随机数。

重复此步骤，直到满足(4)的合成数量为止。

2.2 机器学习模型

2.2.1 逻辑回归

逻辑回归通过函数方法对取值为 0 和 1 的二元响应变量进行建模, 是一种广泛使用且强大的算法^[18]。逻辑回归模型在农业、信用评分、事故分析与预防等方面都具有广泛应用。逻辑回归模型的具体形式为:

$$\pi(x) = \frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}} \quad (7)$$

也可以被写成:

$$g(x) = \ln\left(\frac{\pi(x)}{1 - \pi(x)}\right) = \beta_0 + \beta_1 x \quad (8)$$

2.2.2 随机森林

随机森林是一种用于分类和回归集成分类方法^[19], 具有高精度和稳定性。随机森林由 Breiman 在 2001 年提出^[20], 它由多个决策树组成, 每棵决策树都是独立于其他树构建的。由于集成中有大量决策树, 随机森林算法减少了对数据的过拟合, 所以一般来说, 随机森林算法优于单个决策树。近年来, 随机森林算法在应用中很受欢迎, 在银行业、股票市场、医药领域等都有广泛应用。

随机森林算法的步骤^[21]:

1. 从 Kaggle 信用卡欺诈数据集的训练数据中随机选取部分样本数据。
2. 为每个随机选取的样本数据构建单独的决策树, 将交易分为欺诈交易和正常交易。
3. 决策树由节点分裂而成, 取信息增益最高的节点作为根节点, 对欺诈交易和正常交易进行分类。
4. 决策树的最终输出基于多数投票, 比如输出 0 表示这些是非欺诈案例。
5. 计算欺诈交易和正常交易的准确度(accuracy)、精确度(precision)、召回率(recall)和 F1 分数(F1 score)。

2.3 不平衡问题的分类评价标准

2.3.1 召回率 (recall)、精确度 (precision) 及 F_1 分数

混淆矩阵(confusion matrix)由四部分组成: 「True, False」(真实情况)、「Positive, Negative」(预测情况)。

表 1 混淆矩阵

Table 1 Confusion matrix

	Positive (预测)	Negative (预测)
True (真实)	True Positive (TP)	True Negative (TN)
False (真实)	False Positive (FP)	False Negative (FN)

对二分类问题而言, 有准确度(accuracy)、精确度(precision)、召回率(recall)、 F_1 分数几个分类指标。准确度(accuracy)是最常用的分类指标:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (9)$$

准确度(accuracy)表示模型的正确程度, 它计算所有预测结果中, 模型预测正确的样本比例, 需要在平衡数据中使用。当数据不平衡时, 这一指标容易造成误导, 使结果向样本数量较多的一侧偏移。

精确度(precision)计算所有预测为正类的样本中, 真实为正类的比例。

$$Precision = \frac{TP}{TP+FP} \quad (10)$$

召回率(recall)表示所有正确分类占真实为正类的样本的比例。

$$Recall = \frac{TP}{TP+FN} \quad (11)$$

F_1 分数(F_1 score)用于衡量二分类模型精确度, 是精确度(precision)和召回率(recall)的调和平均, 最大值是 1, 最小值是 0, F_1 值越大意味着模型越好。

$$F_1 = \frac{2*Precision*Recall}{Precision+Recall} \quad (12)$$

对于信用卡欺诈问题来说, 精确度(precision)表示模型在检测欺诈交易方面的准确度, 召回率(recall) 表示检测到欺诈交易到百分比, 所以我们希望召回率(recall)可

以尽量高，并且精确度(precision)在召回率(recall)较高的基础上可以尽可能地高^[22]。

2.3.2 ROC 曲线和 AUC 面积

样本的分类基于阈值的大小，而 ROC 曲线是在各种阈值设置下对分类问题的性能度量。ROC 曲线本质是概率曲线，曲线下面积(AUC)越高，模型在预测欺诈/非欺诈交易方面的效果就越好。ROC 曲线的横轴为 FPR（假正率），纵轴为 TPR（真正率/召回率），计算公式如下：

$$TPR/Recall/Sensitivity = \frac{TP}{TP+FN} \quad (13)$$

$$Specificity = \frac{TN}{TN+FP} \quad (14)$$

$$FPR = 1 - Specificity = \frac{FP}{TN+FP} \quad (15)$$

一个优秀模型的 AUC 接近于 1，这意味着它可以很好地区分正负两类。较差模型的 AUC 接近 0，这意味着它的可分离性很差。理论上的 ROC 曲线如图 2.1 所示。

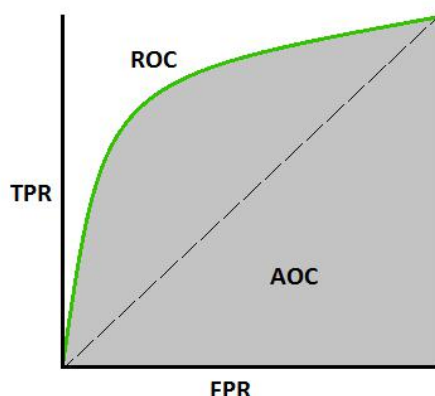


图 2.1 ROC 曲线与 AUC 面积

Figure 2.1 ROC curve and AUC

P-R 曲线是精确度-召回率曲线，即曲线的横轴为召回率(recall)，纵轴为精确度(precision)。一个良好的算法希望在召回率(recall) 较高的基础上获得尽可能地高的精确度(precision)，然而，大多数机器学习算法需要在召回率和精确度之间权衡。在不平衡问题中，ROC 曲线会作出一个比较乐观的估计，而 P-R 曲线则因为精确度(precision)的存在会不断显现 FP 的影响。

3. 数据分析

3.1 数据集介绍

本文采用 Kaggle 中的信用卡数据集，该数据集记录了 2013 年 9 月欧洲一家银行的信用卡持卡人两天内的交易记录数据。数据集共有 284807 条交易记录，且没有缺失值，其中包含 492 条欺诈交易，欺诈交易约占总交易数的 0.172%。该数据含有 30 个特征，其中 V1, V2,... V28 这 28 个特征，最初可能是信用卡号、还款到期日、CVV、持卡人姓名、交易地点、交易日期时间等，但由于隐私考虑，这些特征全部通过 PCA 方法变换。另外两个特征“Time”和“Amount”未经 PCA 变换，其中“Time”是每个交易与数据集中第一个交易之间经过的秒数，特征“Amount”表示交易金额。数据集的响应变量是“Class”，当为欺诈交易时，Class=1；否则为 0。下图（图 3.1）展示了该数据集中欺诈交易和正常交易的分布，可以发现该数据集高度不平衡，正常交易占绝大多数。

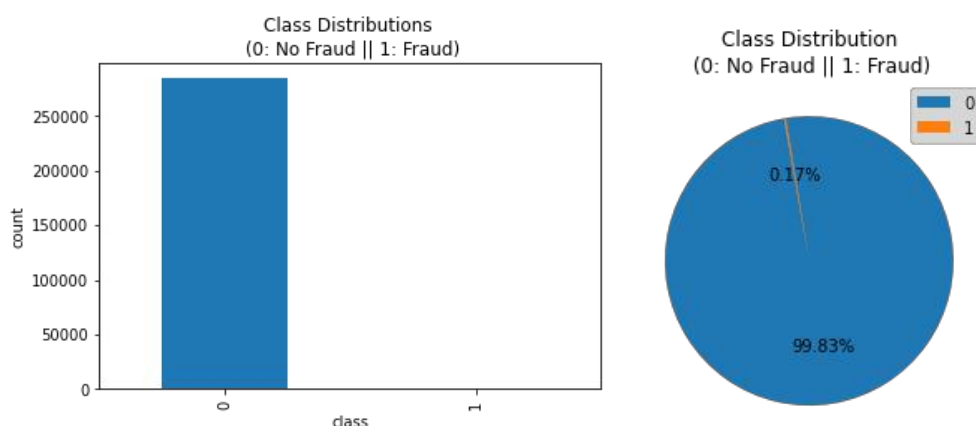


图 3.1 数据集中欺诈交易和正常交易的分布图

Figure 3.1 Fraudulent and Non-Fraudulent Distribution

3.2 探索性数据分析

3.2.1 各变量的相关关系

如果两个特征之间存在高度相关性，那么为避免模型过拟合最好不要纳入模型。由于 V1, V2,... V28 这 28 个特征经过 PCA 变换，所以相互独立，我们只需探究类别 Class，交易金额 Amount 和时间 Time 之间的相关性，如下图(图 3.2)所示：

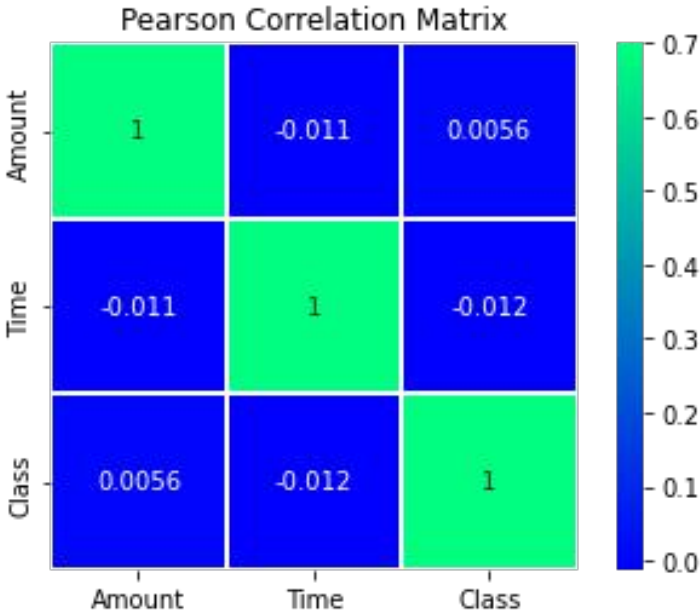


图 3.2 类别 Class，交易金额 Amount 和时间 Time 之间的相关性

Figure 3.1 The correlation between Class, Amount and Time

可见，各变量之间无明显相关性。

3.2.2 数据归一化处理

数据集中包含具有多个范围的数据，比如 V1, V2,... V28 这 28 个特征数据均在 (-1, 1)内，而 “Amount” 数据浮动较大 (图 3.2)。因此，为了避免算法上的不一致，必须把数据进行归一化处理，使数据范围相同，重要程度相当。首先对 “Amount” 特征作箱形图，发现大多数交易金额在 0 到 3000 之间，但对于欺诈交易(Class=0) 存在一些交易金额极大的异常值。

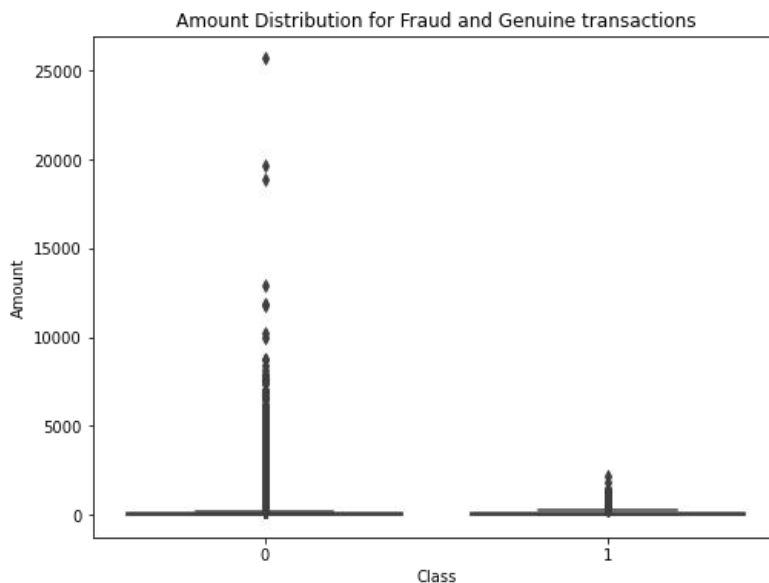


图 3.3 Amount 变量关于欺诈交易和正常交易的分布图

Figure 3.3 Amount Distribution for Fraud and Non-fraud Transactions

下面考虑两种数据归一化方法:

1. log 函数转换

转化函数为: $x^* = \log(x)$ 。考虑到对于欺诈交易存在许多数值极大的异常值, log 函数转换对偏大值的处理效果较好。

2. z-score 标准化

转化函数为: $x^* = \frac{x - \mu}{\sigma}$, 其中 μ 为所有样本数据的均值, σ 为所有样本数据的标准差。

下图 (图 3.3) 展示了两种标准化处理后, Amount 数据的分布情况, 发现 log 函数转换对 Amount 变量中的异常值处理效果较好, 所以使用 log 函数转换后的 Amount 数据进行建模分析。

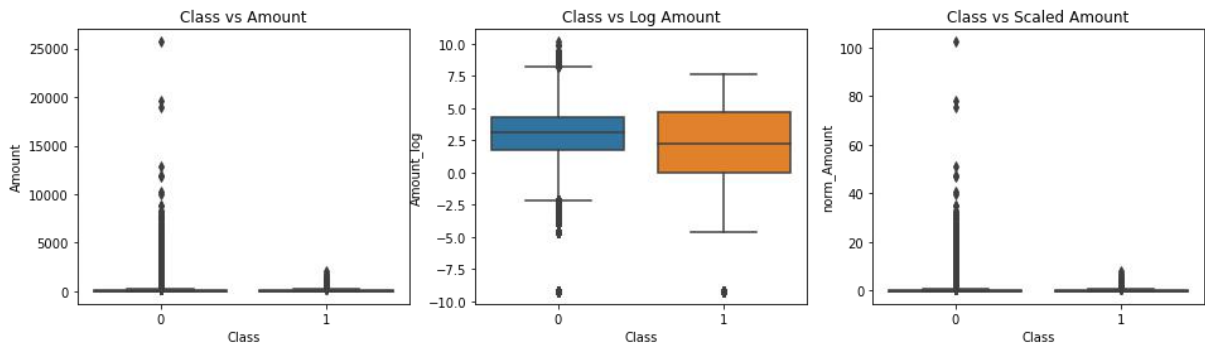


图 3.4 标准化处理前后 Amount 数据的分布

Figure 3.4 Amount Distribution for Fraud and Non-fraud Transactions

3.2.3 特征选择

特征选择保留了数据集中最相关的变量，去除不太重要的特征以减少过拟合，提高准确性，减少模型训练时间。这里选用 Filter 过滤法中的互信息法进行特征选择，互信息越大，则特征与标签类相互依赖程度越大，两者相关性越强。下表展示了每个特征与目标标签的互信息值(表 2)。从中可见，所有特征的互信息熵均大于 0，所以所以特征都和目标标签有一定相关性，在拟合模型时，我们保留所有特征。

表 2 各个特征与目标标签的互信息值

Table 4 Mutual information between different features and Class

Features	V1	V2	V3	V4	V5	V6	V7	V8	V9
Mutual Information	0.00209	0.00321	0.00512	0.00501	0.00244	0.00249	0.00387	0.00212	0.00423
Features	V10	V11	V12	V13	V14	V15	V16	V17	V18
Mutual Information	0.00731	0.00708	0.00766	0.000463	0.00826	0.00058	0.00606	0.00818	0.0042
Features	V19	V20	V21	V22	V23	V24	V25	V26	V27
Mutual Information	0.00143	0.00112	0.00231	0.00035	0.00065	0.00058	0.00044	0.00055	0.00226
Features	V28	Time	Amount						
Mutual Information	0.00179	0.00179	0.00116						

4. 建模分析

训练集和测试通过 70:30 的比例划分，并对训练集的数据进行平衡化处理，然后建立逻辑回归和随机森林模型。模型的参数的设置参考了文献中普遍有效的参数设置，并通过对数据集的初步测试确定。由于参数微调需要消耗大量的时间和精力，还可能导致特定数据集的过拟合问题，所以我们没有对参数进一步微调。虽然对特定数据集参数进行微调可能会提高预测效果，但在实践中更一般的做法是使用普遍有效的参数设置。

4.1 逻辑回归

将数据集应用于逻辑回归模型，不同的平衡数据方式的结果如下：

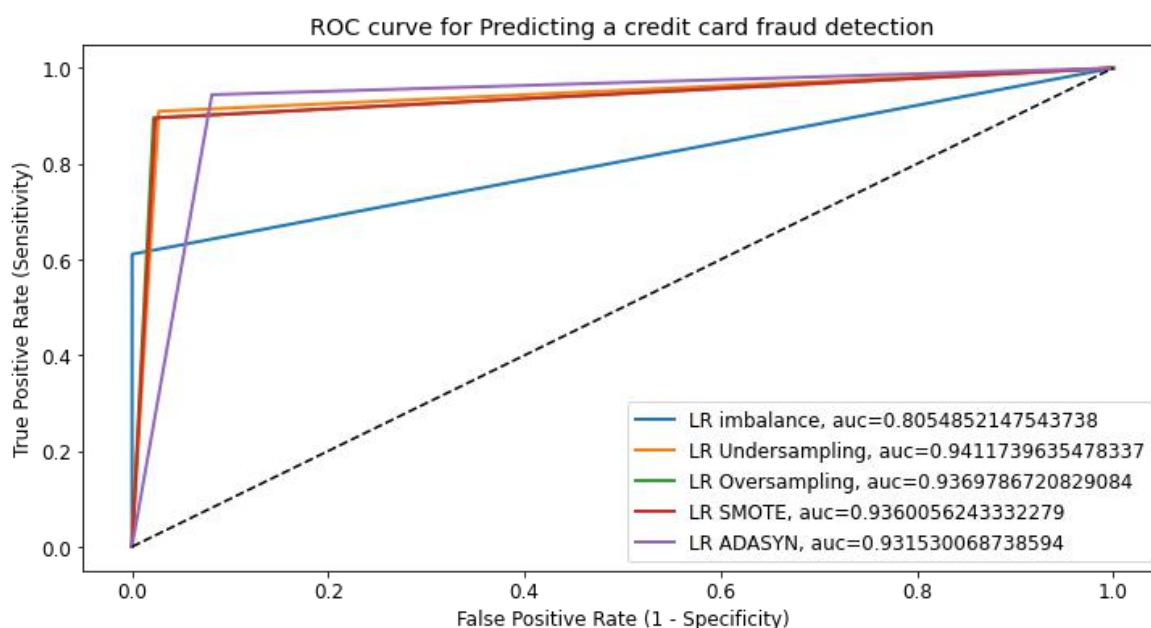


图 4.1 不同逻辑回归模型的 ROC 曲线

Figure 4.1 ROC curve for different Linear Regression models

4.2 随机森林

将数据集应用于随机森林模型，不同的平衡数据方式的结果如下：

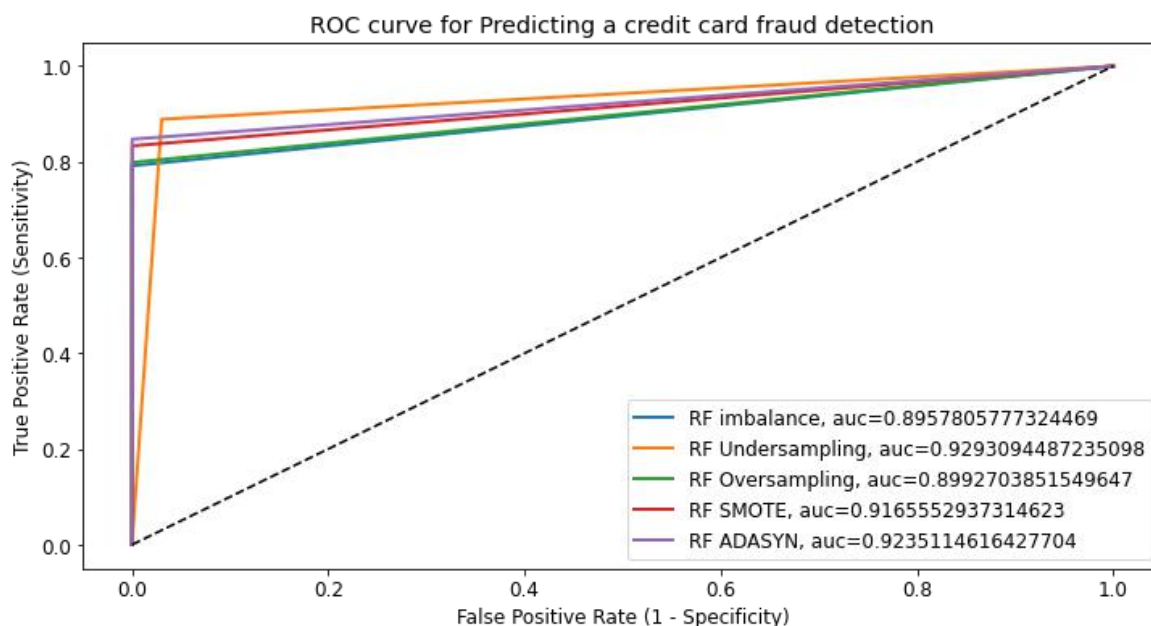


图 4.2 不同随机森林模型的 ROC 曲线

Figure 4.2 ROC curve for different Random Forest models

4.3 结果分析

如下表 (表 3) 所示, 对不平衡数据集分别使用随机过采样、随机欠采样、SMOTE 过采样和自适应合成采样(ADASYN)进行处理, 并且使用逻辑回归和随机森林算法训练模型进行分类, 得出的各项评估指标值。其中, 准确度(accuracy) 和 AUC 不太适合作为不平衡数据的评估指标, 因此我们关注 F1 score 来比较各个模型的性能。

从结果可以发现, 每一种算法的准确度(accuracy)都很高, 但需要将准确度与其他指标结合比较。我们发现随机森林算法整体表现更好, 尤其是 SMOTE+随机森林模型和 ADASYN+随机森林模型具有较高的 TP 值和较低的 FP 值, 在召回率(recall) 较高的基础上, 也有较高的精确度(precision)。其中, 随机过采样+随机森林模型具有最高的 F1 分数, 为 0.8679。可见, 对数据进行平衡化处理, 可以提高模型的召回率(recall), 即提高所有 class 为 1 的样本中正确预测的比例。逻辑回归在检测信用卡欺诈问题上表现欠佳, 即使数据经过平衡化处理, 也不能保证模型在拥有较高召回率(recall)的同时, 也保持较高的精确度(precision)。

因此, 使用过采样技术的随机森林模型是最为理想的模型。另外, 从运行耗时的角度 (表 4), 使用过采样技术的随机森林模型的运行耗时也短于 SMOTE+随机森林

模型和 ADASYN+随机森林模型。

表 3 不同信用卡欺诈模型的表现比较

Table 3 Performance measures of various classifiers

	Model	Accuracy	AUC	Precision	Recall	F ₁ score
7	RF Oversampling	0.99959	0.899270	0.950413	0.798611	0.867925
9	RF ADASYN	0.99954	0.923511	0.877698	0.847222	0.862191
5	RF imbalance	0.99954	0.895781	0.926829	0.791667	0.853933
8	RF SMOTE	0.99950	0.916555	0.863309	0.833333	0.848057
0	LR imbalance	0.99920	0.805485	0.880000	0.611111	0.721311
2	LR Oversampling	0.97799	0.936979	0.064662	0.895833	0.120617
3	LR SMOTE	0.97604	0.936006	0.059695	0.895833	0.111931
1	LR Undersampling	0.97252	0.941174	0.053122	0.909722	0.100383
6	RF Undersampling	0.96959	0.929309	0.047232	0.888889	0.089699
4	LR ADASYN	0.91866	0.931530	0.019214	0.944444	0.037663

表 4 不同信用卡欺诈模型运行耗时比较

Table 4 Runnnig time of different classifiers

	imbalance	Undersampling	Oversampling	SMOTE	ADASYN
LR/s	0.779	0.963	2.283	3.745	5.752
RF/s	112.635	113.358	199.524	391.926	604.518

5. 结论与展望

本文探究了逻辑回归和随机森林在检测信用卡欺诈问题中的性能, 并使用不同的方法处理不平衡数据集。我们使用了 2013 年 9 月欧洲一家银行的信用卡持卡人两天内的交易记录数据集。随机森林算法是近年来备受瞩目的机器学习算法, 它在一系列应用中都展现了卓越的性能。但在信用卡欺诈检测问题上, 随机森林算法往往由于数据不平衡问题受到很多限制, 导致检测效果不佳。本文把随机欠采样、随机过采样、SMOTE 过采样和自适应合成采样 ADASYN 与随机森林算法结合, 对随机森林算法进行改进。对通过建模对比各个模型的 F_1 score, 我们发现:

1. 随机森林+随机过采样在信用卡欺诈检测问题上表现出很好的效果, 在召回率(recall)较高的同时, 达到了最高的精确度(precision)。
2. 从实际使用的角度来看, 随机森林+随机过采样算法运行耗时相比于随机森林+SMOTE 和随机森林+ADASYN 减少很多, 计算效率高, 而且只有两个可调参数, 并在应用中可设置为默认值。
3. 一直以来, 逻辑回归作为许多数据挖掘应用程序中的标准技术有着广泛使用, 但在检测信用卡欺诈问题中没有表现出最佳性能。

本文也存在着许多不足, 比如:

1. 本文使用的 Kaggle 上的 2013 年 9 月欧洲一家银行的信用卡数据集时效性较差, 而且由于信用卡隐私问题, 除了时间 Time 和交易金额 Amount 两个特征, 其余特征均经过 PCA 处理, 数据的可解释性较差。
2. 本文没有刻意优化技术参数, 调整参数后的模型效果可能会优于现有模型。
3. 由于时间精力有限, 本文没有使用当下流行的支持向量机 SVM 和神经网络模型。

今后的研究要侧重于不同机器学习算法, 比如深度学习算法, 来更准确地检测信用卡欺诈问题。另外, 可以探索如何区分具有多次欺诈交易的信用卡和具有少数欺诈交易的信用卡, 利用机器学习方法检测出不同欺诈行为的差异。随着信用卡诈骗犯技术的更新, 信用卡欺诈检测技术也要提高, 信用卡欺诈风险管理需要多方共同合作。

参考文献

- [1] 王雅琼. 信用卡诈骗罪研究[D]. 上海: 华东政法大学, 2013.
- [2] 蒋亚平, 杨丰君. 浅谈数据挖掘在银行信用卡管理中的应用[J]. 时代报告(学术版), 2012(12): 42.
- [3] R. J. Bolton and D. J. Hand, Unsupervised profiling methods for fraud detection Conference on Credit Scoring and Credit Control, 2001.
- [4] Ghosh, Sushmito, and Douglas L. Reilly. "Credit card fraud detection with a neural-network." System Sciences, 1994. Proceedings of the Twenty-Seventh Hawaii International Conference on. Vol. 3. IEEE, 1994.
- [5] Srivastava, Abhinav, et al. "Credit card fraud detection using hidden Markov model." IEEE Transactions on dependable and secure computing 5.1 (2008): 37-48.
- [6] Sahin, Y., and Ekrem Duman. "Detecting credit card fraud by decision trees and support vector machines." World Congress on Engineering 2012. July 4-6, 2012. London, UK.. Vol. 2188. International Association of Engineers, 2010.
- [7] Mishra, Ankit, and Chaitanya Ghorpade. "Credit card fraud detection on the skewed data using various classification and ensemble techniques." 2018 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS). IEEE, 2018: 1-5.
- [8] Xuan, Shiyang, et al. "Random forest for credit card fraud detection." 2018 IEEE 15th international conference on networking, sensing and control (ICNSC). IEEE, 2018.
- [9] 吴婷. 数据挖掘在信用卡欺诈识别上的应用研究[D]. 南京: 东南大学, 2006.
- [10] 杨玺. 基于支持向量机的信用卡欺诈检测研究[D]. 硕士学位论文]. 成都: 四川师范大学, 2008.
- [11] 刘艳红. SVDD 算法研究及在信用卡欺诈检测中的应用[D]. 硕士学位论文]. 镇江: 江苏大学, 2010.
- [12] 陈冠宇. 基于 kNN-Smote-LSTM 的信用卡欺诈风险检测网络模型[D]. 浙江工商大学, 2018.
- [13] 琚春华, 陈冠宇, 鲍福光. 基于 kNN-Smote-LSTM 的消费金融风险检测模型[J]. 系统科学与数学, 2021, 41(2): 481.
- [14] 黄勇新. 基于深度森林的信用卡欺诈检测研究[D]. 暨南大学, 2020.

- [15] Chawla N V, Bowyer K W, Hall L O, et al. SMOTE: synthetic minority over-sampling technique[J]. Journal of artificial intelligence research, 2002, 16: 321-357.
- [16] Brandt J, Lanzén E. A Comparative Review of SMOTE and ADASYN in Imbalanced Data Classification[J]. 2021.
- [17] He H, Bai Y, Garcia E A, et al. ADASYN: Adaptive synthetic sampling approach for imbalanced learning[C]//2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence). IEEE, 2008: 1322-1328.
- [18] Hosmer Jr D W, Lemeshow S, Sturdivant R X. Applied logistic regression[M]. John Wiley & Sons, 2013.
- [19] 王天健. 基于随机森林的信用卡欺诈检测研究[D]. 哈尔滨理工大学, 2020.
- [20] Breiman L. Random forests[J]. Machine Learning, 2001, 45(1): 5-32.
- [21] R. Sailusha, V. Gnaneswar, R. Ramesh and G. R. Rao, "Credit Card Fraud Detection Using Machine Learning," 2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS), 2020, pp. 1264-1270
- [22] 谢声和. 信用卡欺诈检测的机器学习方法比较[D]. 华中师范大学, 2020.

附录

```
1. import matplotlib.pyplot as plt
2. import numpy as np
3. import seaborn as sns
4. from sklearn.preprocessing import StandardScaler
5. from sklearn.model_selection import StratifiedShuffleSplit
6. from sklearn.ensemble import RandomForestClassifier
7. from sklearn.linear_model import LogisticRegression
8. from sklearn.model_selection import train_test_split
9. from sklearn import metrics
10. from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_
    score, accuracy_score, classification_report
11. from imblearn.over_sampling import SMOTE, ADASYN
12. from imblearn.under_sampling import RandomUnderSampler
13. import matplotlib.gridspec as gridspec
14.
15. %matplotlib inline
16.
17. my_data=pd.read_csv("creditcard.csv")
18. my_data.isnull().sum().max()
19.
20. # 数据分布柱状图
21. count_class=pd.value_counts(my_data["Class"],sort=True).sort_index()
22. count_class.plot(kind="bar")
23. plt.title('Class Distributions \n (0: No Fraud || 1: Fraud)')
24. plt.xlabel("class")
25. plt.ylabel("count")
26.
27. # 数据分布饼图
28. explode = [0,0.001]
29. plt.pie(my_data['Class'].value_counts(),explode=explode,autopct='%1.2f%%',
    startangle=100)
30. plt.legend(labels=['0','1'])
31. plt.title('Class Distribution\n (0: No Fraud || 1: Fraud)');
32.
33. print(my_data['Class'].value_counts())
```

```
34. print()
35. print('No Frauds',round(my_data['Class'].value_counts()[0]/len(my_data) * 100,2), '% of the dataset')
36. print('Frauds', round(my_data['Class'].value_counts()[1]/len(my_data) * 100,2), '% of the dataset')
37.
38. #描述性统计数据
39. my_data[['Time','Amount']].describe()
40.
41. # 分布
42. plt.figure(figsize=(8,6))
43. plt.title('Distribution of Transaction Amount', fontsize=14)
44. sns.distplot(my_data['Amount'], bins=100)
45. plt.show()
46.
47. fig, axs = plt.subplots(ncols=2,figsize=(16,4))
48. sns.distplot(my_data[my_data['Class'] == 1]['Amount'], bins=100, ax=axs[0])
49. axs[0].set_title("Distribution of Fraud Transactions")
50. sns.distplot(my_data[my_data['Class'] == 0]['Amount'], bins=100, ax=axs[1])
51. axs[1].set_title("Distribution of Genuine Transactions")
52. plt.show()
53.
54. print("Fraud Transaction distribution : \n",my_data[my_data['Class'] == 1]['Amount'].value_counts().head())
55. print("\n")
56. print("Maximum amount of fraud transaction - ",my_data[my_data['Class'] == 1]['Amount'].max())
57. print("Minimum amount of fraud transaction - ",my_data[my_data['Class'] == 1]['Amount'].min())
58.
59. plt.figure(figsize=(8,6))
60. sns.boxplot(x='Class', y='Amount',data = my_data)
61. plt.title('Amount Distribution for Fraud and Genuine transactions')
62. plt.show()
63.
64. #是否有异常值
65. plt.figure(figsize=(8,6))
```

```
66. plt.title('Distribution of Transaction Time', fontsize=14)
67. sns.distplot(my_data['Time'], bins=100)
68. plt.show()
69.
70. fig, axs = plt.subplots(ncols=2, figsize=(16,4))
71.
72. sns.distplot(my_data[(my_data['Class'] == 1)]['Time'], bins=100, color='red', a
x=axs[0])
73. axs[0].set_title("Distribution of Fraud Transactions")
74. sns.distplot(my_data[(my_data['Class'] == 0)]['Time'], bins=100, color='green',
ax=axs[1])
75. axs[1].set_title("Distribution of Genuine Transactions")
76. plt.show()
77.
78. #检查数据相关性
79. my_data[['Amount', 'Time', 'Class']].corr()['Class'].sort_values( ascending=False).head(10)
80. plt.title('Pearson Correlation Matrix')
81. sns.heatmap(my_data[['Amount', 'Time', 'Class']].corr(),linewidths=0.25,vmax=
0.7,square=True,cmap="winter",linecolor='w',annot=True)
82.
83. # 数据归一化
84. # log 函数转换
85. my_data['Amount_log'] = np.log(my_data.Amount + 0.0001)
86. my_data['Time_log'] = np.log(my_data.Time + 0.0001)
87.
88. # 数据标准化
89. my_data["norm_Amount"]=StandardScaler().fit_transform(my_data["Amount"].
values.reshape(-1,1))
90. my_data["norm_Time"]=StandardScaler().fit_transform(my_data["Time"].valu
es.reshape(-1,1))
91.
92. # 比较两种方法好坏
93. fig , axs = plt.subplots(nrows = 1 , ncols = 3, figsize = (16,4))
94. sns.boxplot(x ="Class",y="Amount",data=my_data, ax = axs[0])
95. axs[0].set_title("Class vs Amount")
96. sns.boxplot(x ="Class",y="Amount_log",data=my_data, ax = axs[1])
```

```
97. axs[1].set_title("Class vs Log Amount")
98. sns.boxplot(x="Class",y="norm_Amount",data=my_data, ax = axs[2])
99. axs[2].set_title("Class vs Scaled Amount")
100.
101. fig , axs = plt.subplots(nrows = 1 , ncols = 3, figsize = (16,4))
102. sns.boxplot(x="Class",y="Time",data=my_data, ax = axs[0])
103. axs[0].set_title("Class vs Time")
104. sns.boxplot(x="Class",y="Time_log",data=my_data, ax = axs[1])
105. axs[1].set_title("Class vs Log Time")
106. sns.boxplot(x="Class",y="norm_Time",data=my_data, ax = axs[2])
107. axs[2].set_title("Class vs Scaled Time")
108.
109. X = my_data.drop(["Time","Amount",'Time_log', 'norm_Amount', 'norm_Time', "Class"],axis=1)
110. y = my_data['Class']
111. # 划分训练集和测试集
112. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, shuffle=True, random_state=101)
113. print("X_train - ",X_train.shape)
114. print("y_train - ",y_train.shape)
115. print("X_test - ",X_test.shape)
116. print("y_test - ",y_test.shape)
117.
118. #逻辑回归
119. logreg = LogisticRegression()
120. logreg.fit(X_train, y_train)
121. y_pred = logreg.predict(X_test)
122.
123. print('Accuracy:{0:0.5f}'.format(metrics.accuracy_score(y_pred, y_test)))
124. print('AUC:{0:0.5f}'.format(metrics.roc_auc_score(y_test,y_pred)))
125. print('Precision:{0:0.5f}'.format(metrics.precision_score(y_test, y_pred)))
126. print('Recall:{0:0.5f}'.format(metrics.recall_score(y_test, y_pred)))
127. print('F1:{0:0.5f}'.format(metrics.f1_score(y_test,y_pred)))
128. print(metrics.classification_report(y_test, y_pred))
129.
130. print(pd.Series(y_test).value_counts())
131. print()
```

```
132.   cnf_matrix = metrics.confusion_matrix(y_test,y_pred)
133.   print(cnf_matrix)
134.
135.   # 混淆矩阵
136.   p = sns.heatmap(pd.DataFrame(cnf_matrix), annot=True , annot_kws={"size":16},cmap="winter" ,fmt='g')
137.   plt.title('Confusion matrix', y=1.1, fontsize = 18)
138.   plt.ylabel('Actual',fontsize = 16)
139.   plt.xlabel('Predicted',fontsize = 16)
140.   plt.show()
141.
142.   # ROC,AUC
143.   metrics.roc_auc_score(y_test , y_pred)
144.   # ROC 曲线
145.   plt.figure(figsize=(8,6))
146.   fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred)
147.   auc = metrics.roc_auc_score(y_test, y_pred)
148.   print("AUC - ",auc,"\n")
149.   plt.plot(fpr,tpr,linewidth=2, label="data 1, auc="+str(auc))
150.   plt.legend(loc=4)
151.   plt.plot([0,1], [0,1], 'k--' )
152.   plt.rcParams['font.size'] = 12
153.   plt.title('ROC curve for Predicting a credit card fraud detection')
154.   plt.xlabel('False Positive Rate (1 - Specificity)')
155.   plt.ylabel('True Positive Rate (Sensitivity)')
156.   plt.show()
157.
158.   # precision-recall 曲线
159.   precision,recall,thresholds = metrics.precision_recall_curve(y_test, y_pred)
160.   # F1 score
161.   f1 = metrics.f1_score(y_test, y_pred)
162.   print('f1=%0.3f' % (f1))
163.   # plot no skill
164.   plt.plot([0, 1], [0.5, 0.5], linestyle='--')
165.   # plot the roc curve for the model
166.   plt.plot(recall, precision, marker='.')
```

```
167.     # show the plot
168.     plt.show()
169.
170.     # 不平衡数据的处理
171.     # 随机过采样 & 随机欠采样
172.     # 逻辑回归 + 随机欠采样
173.     from collections import Counter
174.     from sklearn.datasets import make_classification
175.     rus = RandomUnderSampler(random_state=42)
176.     X_train_rus, y_train_rus = rus.fit_resample(X_train, y_train)
177.     print('Original dataset shape %s' % Counter(y_train))
178.     print('Resampled dataset shape %s' % Counter(y_train_rus))
179.     logreg = LogisticRegression()
180.     logreg.fit(X_train_rus, y_train_rus)
181.     y_pred_rus = logreg.predict(X_test)
182.     print('Accuracy :{0:0.5f}'.format(metrics.accuracy_score(y_pred_rus, y_test)))
183.     print('AUC:{0:0.5f}'.format(metrics.roc_auc_score(y_test, y_pred_rus)))
184.     print('Precision:{0:0.5f}'.format(metrics.precision_score(y_test,y_pred_rus)))
185.     print('Recall:{0:0.5f}'.format(metrics.recall_score(y_test, y_pred_rus)))
186.     print('F1:{0:0.5f}'.format(metrics.f1_score(y_test,y_pred_rus)))
187.     # ROC 曲线
188.     plt.figure(figsize=(8,6))
189.     fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_rus)
190.     auc = metrics.roc_auc_score(y_test, y_pred_rus)
191.     print("AUC - ",auc,"\n")
192.     plt.plot(fpr,tpr,linewidth=2, label="data 1, auc="+str(auc))
193.     plt.legend(loc=4)
194.     plt.plot([0,1], [0,1], 'k--' )
195.     plt.rcParams['font.size'] = 12
196.     plt.title('ROC curve for Predicting a credit card fraud detection')
197.     plt.xlabel('False Positive Rate (1 - Specificity)')
198.     plt.ylabel('True Positive Rate (Sensitivity)')
199.     plt.show()
200.     # precision-recall curve
```



```
201. precision,recall,thresholds = metrics.precision_recall_curve(y_test,y_pred
_rus)
202. # F1 score
203. f1 = metrics.f1_score(y_test, y_pred_rus)
204. print('f1=%.3f' % (f1))
205. plt.plot([0, 1], [0.5, 0.5], linestyle='--')
206. plt.plot(recall, precision, marker='.')
207. plt.show()
208. # 混淆矩阵
209. cnf_matrix = metrics.confusion_matrix(y_test , y_pred_rus)
210. sns.heatmap(pd.DataFrame(cnf_matrix),annot=True, annot_kws={"size":
16}, cmap="winter" ,fmt='g')
211. plt.title('Confusion matrix', y=1.1, fontsize = 18)
212. plt.xlabel('Predicted',fontsize = 16)
213. plt.ylabel('Actual',fontsize = 16)
214. plt.show()
215.
216. # 逻辑回归 + 随机过采样
217. from imblearn.over_sampling import RandomOverSampler
218. ros = RandomOverSampler(random_state=42)
219. X_train_ros, y_train_ros = ros.fit_resample(X_train, y_train)
220. print('Original dataset shape %s' % Counter(y_train))
221. print('Resampled dataset shape %s' % Counter(y_train_ros))
222. logreg = LogisticRegression()
223. logreg.fit(X_train_ros, y_train_ros)
224. y_pred_ros = logreg.predict(X_test)
225. print('Accuracy :{0:0.5f}'.format(metrics.accuracy_score(y_pred_ros,y_te
st)))
226. print('AUC:{0:0.5f}'.format(metrics.roc_auc_score(y_test, y_pred_ros)))
227. print('Precision:{0:0.5f}'.format(metrics.precision_score(y_test ,y_pred_r
os)))
228. print('Recall:{0:0.5f}'.format(metrics.recall_score(y_test, y_pred_ros)))
229. print('F1:{0:0.5f}'.format(metrics.f1_score(y_test ,y_pred_ros)))
230. # ROC 曲线
231. plt.figure(figsize=(8,6))
232. fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_ros)
233. auc = metrics.roc_auc_score(y_test, y_pred_ros)
```

```
234.     print("AUC - ",auc,"\n")
235.     plt.plot(fpr, tpr, linewidth=2, label="data 1, auc="+str(auc))
236.     plt.legend(loc=4)
237.     plt.plot([0,1], [0,1], 'k--' )
238.     plt.rcParams['font.size'] = 12
239.     plt.title('ROC curve for Predicting a credit card fraud detection')
240.     # precision-recall 曲线
241.     precision, recall, thresholds = metrics.precision_recall_curve(y_test, y_pred_ros)
242.     # F1 score
243.     f1 = metrics.f1_score(y_test, y_pred_ros)
244.     print('f1=%.3f' % (f1))
245.     plt.plot([0, 1], [0.5, 0.5], linestyle='--')
246.     plt.plot(recall, precision, marker='.')
247.     plt.show()
248.     plt.xlabel('False Positive Rate (1 - Specificity)')
249.     plt.ylabel('True Positive Rate (Sensitivity)')
250.     plt.show()
251.     # 混淆矩阵
252.     cnf_matrix = metrics.confusion_matrix(y_test, y_pred_ros)
253.     sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, annot_kws={"size":
16}, cmap="winter", fmt='g')
254.     plt.title('Confusion matrix', y=1.1, fontsize = 18)
255.     plt.xlabel('Predicted', fontsize = 16)
256.     plt.ylabel('Actual', fontsize = 16)
257.     plt.show()
258.
259.     # 逻辑回归 + SMOTE
260.     smote = SMOTE(random_state=42)
261.     X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
262.     print('Original dataset shape %s' % Counter(y_train))
263.     print('Resampled dataset shape %s' % Counter(y_train_smote))
264.     logreg = LogisticRegression(max_iter=1000)
265.     logreg.fit(X_train_smote, y_train_smote)
266.     y_pred_smote = logreg.predict(X_test)
267.     print('Accuracy:{0:0.5f}'.format(metrics.accuracy_score(y_test, y_pred_smote)))
```

```
268.     print('AUC:{0:0.5f}'.format(metrics.roc_auc_score(y_test , y_pred_smote))
269.     print('Precision:{0:0.5f}'.format(metrics.precision_score(y_test , y_pred_s
270.     print('Recall:{0:0.5f}'.format(metrics.recall_score(y_test , y_pred_smote)))
271.     print('F1:{0:0.5f}'.format(metrics.f1_score(y_test , y_pred_smote)))
272.
273.     plt.figure(figsize=(8,6))
274.     fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_smote)
275.     auc = metrics.roc_auc_score(y_test, y_pred_smote)
276.     print("AUC - ",auc,"\n")
277.     plt.plot(fpr,tpr,linewidth=2, label="data 1, auc="+str(auc))
278.     plt.legend(loc=4)
279.     plt.plot([0,1], [0,1], 'k--' )
280.     plt.rcParams['font.size'] = 12
281.     plt.title('ROC curve for Predicting a credit card fraud detection')
282.     plt.xlabel('False Positive Rate (1 - Specificity)')
283.     plt.ylabel('True Positive Rate (Sensitivity)')
284.
285.     plt.show()
286.     precision, recall, thresholds = metrics.precision_recall_curve(y_test, y_pr
287.     f1 = metrics.f1_score(y_test, y_pred_smote)
288.     print('f1=%0.3f' % (f1))
289.     plt.plot([0, 1], [0.5, 0.5], linestyle='--')
290.     plt.plot(recall, precision, marker='.')
291.     plt.show()
292.
293.     cnf_matrix = metrics.confusion_matrix(y_test , y_pred_smote)
294.     sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, annot_kws={"size":
295.     plt.title('Confusion matrix', y=1.1, fontsize = 18)
296.     plt.xlabel('Predicted',fontsize = 16)
297.     plt.ylabel('Actual',fontsize = 16)
298.     plt.show()
299.
300.     # 逻辑回归 + ADASYN 自适应合成采样
```

```
301. adasyn = ADASYN(random_state=42)
302. X_train_adasyn, y_train_adasyn = adasyn.fit_resample(X_train,y_train)
303. print('Original dataset shape %s' % Counter(y_train))
304. print('Resampled dataset shape %s' % Counter(y_train_adasyn))
305. logreg = LogisticRegression()
306. logreg.fit(X_train_adasyn, y_train_adasyn)
307. y_pred_adasyn = logreg.predict(X_test)
308. print('Accuracy:{0:0.5f}'.format(metrics.accuracy_score(y_pred , y_pred_
    adasyn)))
309. print('AUC:{0:0.5f}'.format(metrics.roc_auc_score(y_test , y_pred_adasyn)
    ))
310. print('Precision:{0:0.5f}'.format(metrics.precision_score(y_test , y_pred_a
    dasyn)))
311. print('Recall:{0:0.5f}'.format(metrics.recall_score(y_test , y_pred_adasyn)
    )
312. print('F1:{0:0.5f}'.format(metrics.f1_score(y_test , y_pred_adasyn)))
313.
314. plt.figure(figsize=(8,6))
315. fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_adasyn)
316. auc = metrics.roc_auc_score(y_test, y_pred_adasyn)
317. print("AUC - ",auc,"\n")
318. plt.plot(fpr,tpr,linewidth=2, label="data 1, auc="+str(auc))
319. plt.legend(loc=4)
320. plt.plot([0,1], [0,1], 'k--' )
321. plt.rcParams['font.size'] = 12
322. plt.title('ROC curve for Predicting a credit card fraud detection')
323. plt.xlabel('False Positive Rate (1 - Specificity)')
324. plt.ylabel('True Positive Rate (Sensitivity)')
325. plt.show()
326.
327. precision, recall, thresholds = metrics.precision_recall_curve(y_test, y_pr
    ed_adasyn)
328. f1 = metrics.f1_score(y_test, y_pred_adasyn)
329. print('f1=%0.3f' % (f1))
330. plt.plot([0, 1], [0.5, 0.5], linestyle='--')
331. plt.plot(recall, precision, marker='.')
332. plt.show()
```

```
333.
334.     cnf_matrix = metrics.confusion_matrix(y_test , y_pred_adasyn)
335.     sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, annot_kws={"size":
336.         16}, cmap="winter" ,fmt='g')
337.     plt.title('Confusion matrix', y=1.1, fontsize = 18)
338.     plt.xlabel('Predicted',fontsize = 16)
339.     plt.ylabel('Actual',fontsize = 16)
340.
341.     names_lst = []
342.
343.     aucs_train_lst = []
344.     accuracy_train_lst = []
345.     precision_train_lst = []
346.     recall_train_lst = []
347.     f1_train_lst = []
348.
349.     aucs_test_lst = []
350.     accuracy_test_lst = []
351.     precision_test_lst = []
352.     recall_test_lst = []
353.     f1_test_lst = []
354.
355.     import time
356.     Time = []
357.     def build_measure_model(models):
358.         plt.figure(figsize=(12,6))
359.         t0=time.time()
360.         for name, model, X_train, y_train, X_test, y_test in models:
361.             names_lst.append(name)
362.
363.             # Buila model
364.             model.fit(X_train, y_train)
365.
366.             # Predict
367.             y_train_pred = model.predict(X_train)
368.             y_test_pred = model.predict(X_test)
```

```
369.         t1=time.time()
370.         Time.append((t1-t0))
371.
372.         # accuracy
373.         Accuracy_train = metrics.accuracy_score(y_train, y_train_pred)
374.         accuracy_train_lst.append(Accuracy_train)
375.         Accuracy_test = metrics.accuracy_score(y_test, y_test_pred)
376.         accuracy_test_lst.append(Accuracy_test)
377.
378.         # auc
379.         Aucs_train = metrics.roc_auc_score(y_train, y_train_pred)
380.         aucs_train_lst.append(Aucs_train)
381.         Aucs_test = metrics.roc_auc_score(y_test , y_test_pred)
382.         aucs_test_lst.append(Aucs_test)
383.
384.         # precision
385.         PrecisionScore_train = metrics.precision_score(y_train , y_train_pred)
386.         precision_train_lst.append(PrecisionScore_train)
387.         PrecisionScore_test = metrics.precision_score(y_test , y_test_pred)
388.         precision_test_lst.append(PrecisionScore_test)
389.
390.         # recall
391.         RecallScore_train = metrics.recall_score(y_train , y_train_pred)
392.         recall_train_lst.append(RecallScore_train)
393.         RecallScore_test = metrics.recall_score(y_test , y_test_pred)
394.         recall_test_lst.append(RecallScore_test)
395.
396.         # F1 score
397.         F1Score_train = metrics.f1_score(y_train , y_train_pred)
398.         f1_train_lst.append(F1Score_train)
399.         F1Score_test = metrics.f1_score(y_test , y_test_pred)
400.         f1_test_lst.append(F1Score_test)
401.
402.         # 混淆矩阵
403.         cnf_matrix = metrics.confusion_matrix(y_test , y_test_pred)
```

```
404.
405.     print("Model Name :", name)
406.     print('Train Accuracy :{0:0.5f}'.format(Accuracy_train))
407.     print('Test Accuracy :{0:0.5f}'.format(Accuracy_test))
408.     print('Train AUC : {0:0.5f}'.format(Aucs_train))
409.     print('Test AUC : {0:0.5f}'.format(Aucs_test))
410.     print('Train Precision : {0:0.5f}'.format(PrecisionScore_train))
411.     print('Test Precision : {0:0.5f}'.format(PrecisionScore_test))
412.     print('Train Recall : {0:0.5f}'.format(RecallScore_train))
413.     print('Test Recall : {0:0.5f}'.format(RecallScore_test))
414.     print('Train F1 : {0:0.5f}'.format(F1Score_train))
415.     print('Test F1 : {0:0.5f}'.format(F1Score_test))
416.     print('Confusion Matrix : \n', cnf_matrix)
417.     print("\n")
418.     # plot ROC Curve
419.     fpr, tpr, thresholds = metrics.roc_curve(y_test, y_test_pred)
420.     auc = metrics.roc_auc_score(y_test, y_test_pred)
421.     plt.plot(fpr,tpr,linewidth=2, label=name + ", auc="+str(auc))
422.
423.     plt.legend(loc=4)
424.     plt.plot([0,1], [0,1], 'k--' )
425.     plt.rcParams['font.size'] = 12
426.     plt.title('ROC curve for Predicting a credit card fraud detection')
427.     plt.xlabel('False Positive Rate (1 – Specificity)')
428.     plt.ylabel('True Positive Rate (Sensitivity)')
429.     #plt.tight_layout()
430.     #fig.savefig("models.png")
431.     plt.show()
432.
433.     # 逻辑回归
434.     LRmodels = []
435.     LRmodels.append(('LR imbalance', LogisticRegression(solver='liblinear',
multi_class='ovr'), X_train,y_train,X_test,y_test))
436.     LRmodels.append(('LR Undersampling', LogisticRegression(solver='liblin
ear', multi_class='ovr'),X_train_rus,y_train_rus,X_test,y_test))
437.     LRmodels.append(('LR Oversampling', LogisticRegression(solver='liblin
ear', multi_class='ovr'),X_train_ros,y_train_ros,X_test,y_test))
```



```
438. LRmodels.append(('LR SMOTE', LogisticRegression(solver='liblinear', multi_class='ovr'), X_train_smote, y_train_smote, X_test, y_test))
439. LRmodels.append(('LR ADASYN', LogisticRegression(solver='liblinear', multi_class='ovr'), X_train_adasyn, y_train_adasyn, X_test, y_test))
440.
441. build_measure_model(LRmodels)
442.
443. # 随机森林
444. RFmodels = []
445.
446. RFmodels.append(('RF imbalance', RandomForestClassifier(), X_train, y_train, X_test, y_test))
447. RFmodels.append(('RF Undersampling', RandomForestClassifier(), X_train_rus, y_train_rus, X_test, y_test))
448. RFmodels.append(('RF Oversampling', RandomForestClassifier(), X_train_ros, y_train_ros, X_test, y_test))
449. RFmodels.append(('RF SMOTE', RandomForestClassifier(), X_train_smote, y_train_smote, X_test, y_test))
450. RFmodels.append(('RF ADASYN', RandomForestClassifier(), X_train_adasyn, y_train_adasyn, X_test, y_test))
451.
452. build_measure_model(RFmodels)
453. Time
454.
455. data = {'Model': names_lst,
456.         #'Accuracy_Train': accuracy_train_lst,
457.         'Accuracy_Test': accuracy_test_lst,
458.         #'AUC_Train': aucs_train_lst,
459.         'AUC_Test': aucs_test_lst,
460.         #'PrecisionScore_Train': precision_train_lst,
461.         'PrecisionScore_Test': precision_test_lst,
462.         #'RecallScore_Train': recall_train_lst,
463.         'RecallScore_Test': recall_test_lst,
464.         #'F1Score_Train': f1_train_lst,
465.         'F1Score_Test': f1_test_lst}
466.
467. print("Performance measures of various classifiers: \n")
```

```
468. performance_df = pd.DataFrame(data)
```

```
469. performance_df.sort_values(['F1Score_Test','RecallScore_Test','AUC_Test'],ascending=False)
```

致谢

四年前，当我填写高考志愿的时候，我没想到自己回到时候读书。但如今，大学四年的生活一晃而过，当我写完这篇毕业论文的时候，心中如释重负，感慨良多。

对于这次毕业论文的撰写，首先要感谢的是孙蕾老师。她在整个过程中都给予了我充分的帮助和支持。孙蕾老师对我的论文改进提出宝贵的建议，在我遇到困难时尽心地进行指点与解答。在此借论文完成之际，表示由衷的感谢与敬意。

其次要感谢我在华东师范大学同学们，感谢她们四年来的陪伴；我还要感谢我来到威斯康星之后认识的新朋友们，感谢她们让我虽然身在异国他乡，却不曾体会过孤独的滋味。

最后感谢我的父母，她们是我的终极榜样，感谢她们为我付出的一切，无论我做什么她们都无条件支持我，她们是我最初的启明灯，引领我走上科学探索之路。

在这里引用加缪《鼠疫》中的一句话：“我感到有趣的是：为所爱而生，为所爱而死。”

夏嘉琦

2022 年 04 月 24 日