借呗额度欺诈分析

蒋璐煜

2020 年 7 月 4 日

目录

1 项目背景

新的任务我们来做一个蚂蚁金服中借呗额度欺诈的问题，数据集包括了 2015 年 9 月份两天时间内的交易数据，284807 笔交易中，一共有 492 笔是欺诈行为。输入数据一共包括了 28 个特征 V1，V2，……V28 对应的

取值，以及交易时间 Time 和交易金额 Amount。为了保护数据隐私，我们不知道 V1 到 V28 这些特征代表的具体含义，只知道这 28 个特征值是通过 PCA 变换得到的结果。另外字段 Class 代表该笔交易的分类，Class=0 为正常（非欺诈），Class=1 代表欺诈。我们的目标是针对这个数据集构建一个信用卡欺诈分析的分类器。

2 项目实战

2.1 数据集分析

首先观察数据结构：

```
def __init__(self):
    self.__data = pd.read_csv('alipay_huabei_FS1.csv')

def get_shape(self):
    return self.__data.shape
```

可以发现数据集规格为 (284807, 31)，31 维特征中，Class 表示分类结果，Time、Amount 数量级较大，V1-28 为结构化数据。

接着观察分类结果分布：

```
def get_class_values(self):
    return pd.value_counts(self.__data['Class'])
```

Class	Count
0	284315
1	492

可以发现分类结果为 0 和 1 的数据量差距悬殊，后续处理中需要考虑如重新采样、更换算法、扩大数据样本、人工生成样本等方法解决这一问题。

2.2 特征工程

2.2.1 查看交易时间-交易金额分布关系

注：已将 Time 换算为小时，范围为 0-48

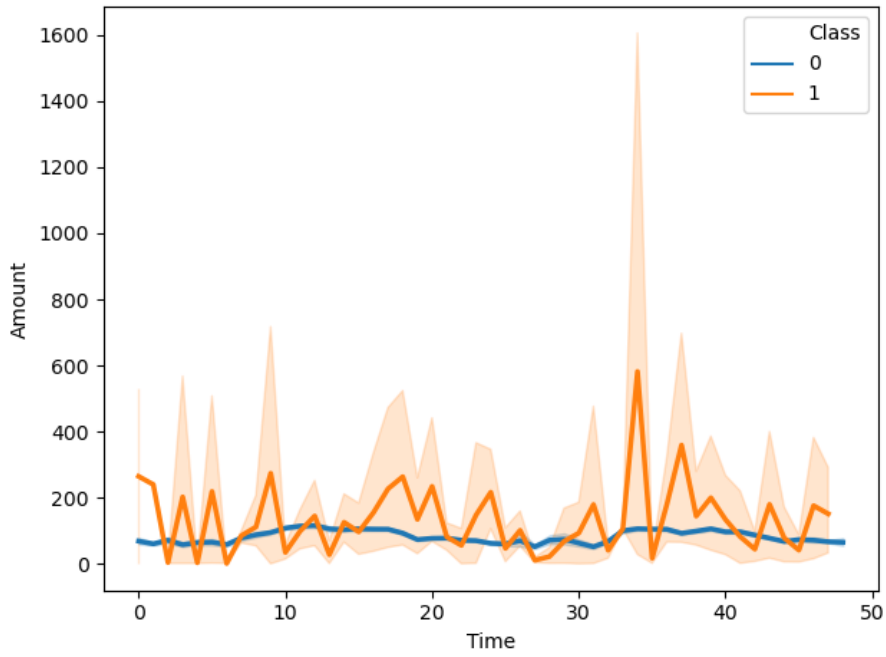

图 1: 交易时间-交易金额分布图

如图 1，可以发现，欺诈数据与正常数据发生的时间与金额分布基本一致，考虑到可能是为了将欺诈交易伪装为正常交易而故意为之的做法。

2.2.2 查看交易金额分布图

如图 2，盗刷的金额与信用卡正常用户发生的金额相比金额较小，猜测是盗刷者为了不引起注意，更倾向于选择小金额消费。

2.2.3 查看交易时间分布图

如图 3，可以发现正常数据与欺诈数据时间分布基本相同。

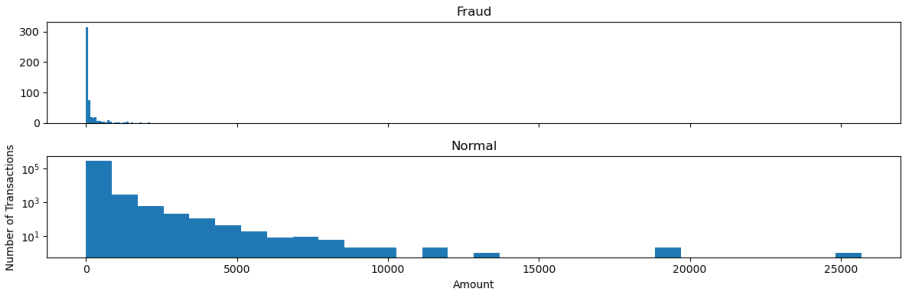

图 2: 交易金额分布图

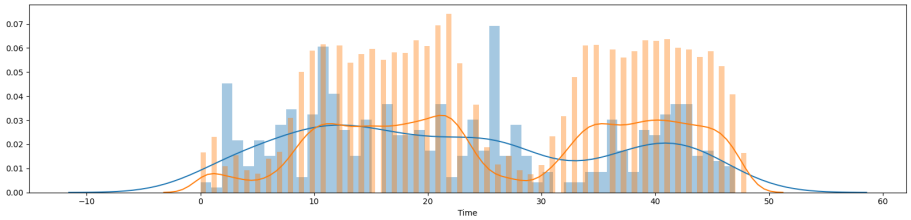

图 3: 交易时间分布图

2.2.4 查看 V1-28 结构化数据分布图

从图中分布可以发现，V8，V13，V15，V20，V21，V22，V23，V24，V25，V26，V27，V28 这些特征中，欺诈与正常数据的分布基本一致，因此对数据分类判断意义不大，可以考虑删去这些特征。

2.3 模型训练

2.3.1 模型选择

该问题是一个二分类问题，数据为结构化数据，这里选用逻辑回归作为分类模型进行训练。此外，由于 V1-28 已经经过 PCA 处理，因此不需要再进行数据降维，后续考虑是否需要特征缩放处理。

2.3.2 数据预处理

解决样本不均衡 这里采用下采样 (under-sampling) 的方法进行处理，通过随机抽取 Class 为 0 的样本使得 Class 为 0 的样本数量与 Class 为 1 的

样本数量达到 1:1，使得训练集内分布均衡。

归一化 Time 列数据由于正常数据与欺诈数据分布基本相同，考虑直接舍去，而 Amount 部分数据较 V1-28 较大，需要做标准化处理：

```
def standardize_amount(self):
    self.__data = self.__raw_data_class.get_data()
    self.__data['StdAmount'] = StandardScaler().fit_transform(
        self.__data['Amount'].values.reshape(-1, 1)
    )
    self.__data = self.__data.drop(['Time', 'Amount'], axis=1)
```

5 折交叉验证 为了辅助调参，这里采用 5 折交叉验证辅助调参，测试集比例 0.3，训练集比例 0.7。

```
def cross_validation_split(self, X, y):
    X_train, X_test, y_train, y_test =
        train_test_split(X, y, test_size=0.3,
                                    random_state=0)
    return X_train, X_test, y_train, y_test
```

2.3.3 训练模型

recall 这里采用召回率进行模型评价。

精度

$$Accuracy = \frac{TP + TN}{total}$$

准确率

$$precision = \frac{TP}{TP + FP}$$

召回率

$$recall = \frac{TP}{TP + FN}$$

训练过程如下：

1. 对模型进行迭代，获取训练结果较好的正则化参数 C

2. 选择最优参数 C 进行建模

3. 绘出混淆矩阵

4. 绘出 ROC 曲线

训练结果：

Best model parameters: C parameter = 0.01

该模型在重采样数据上获得了 0.9252 的召回率，在原数据上获得了 0.9184 的召回率。

混淆矩阵如图 10-11。

ROC 曲线如图 12。

2.3.4 对比训练

在原数据集上直接进行训练:

Best model parameters: C parameter = 1.0

该模型在原数据上获得了 0.5986 的召回率，可以看出效果很差。

混淆矩阵如图 13。

ROC 曲线如图 14。

2.3.5 预测概率

混淆矩阵如图 15。

3 项目总结

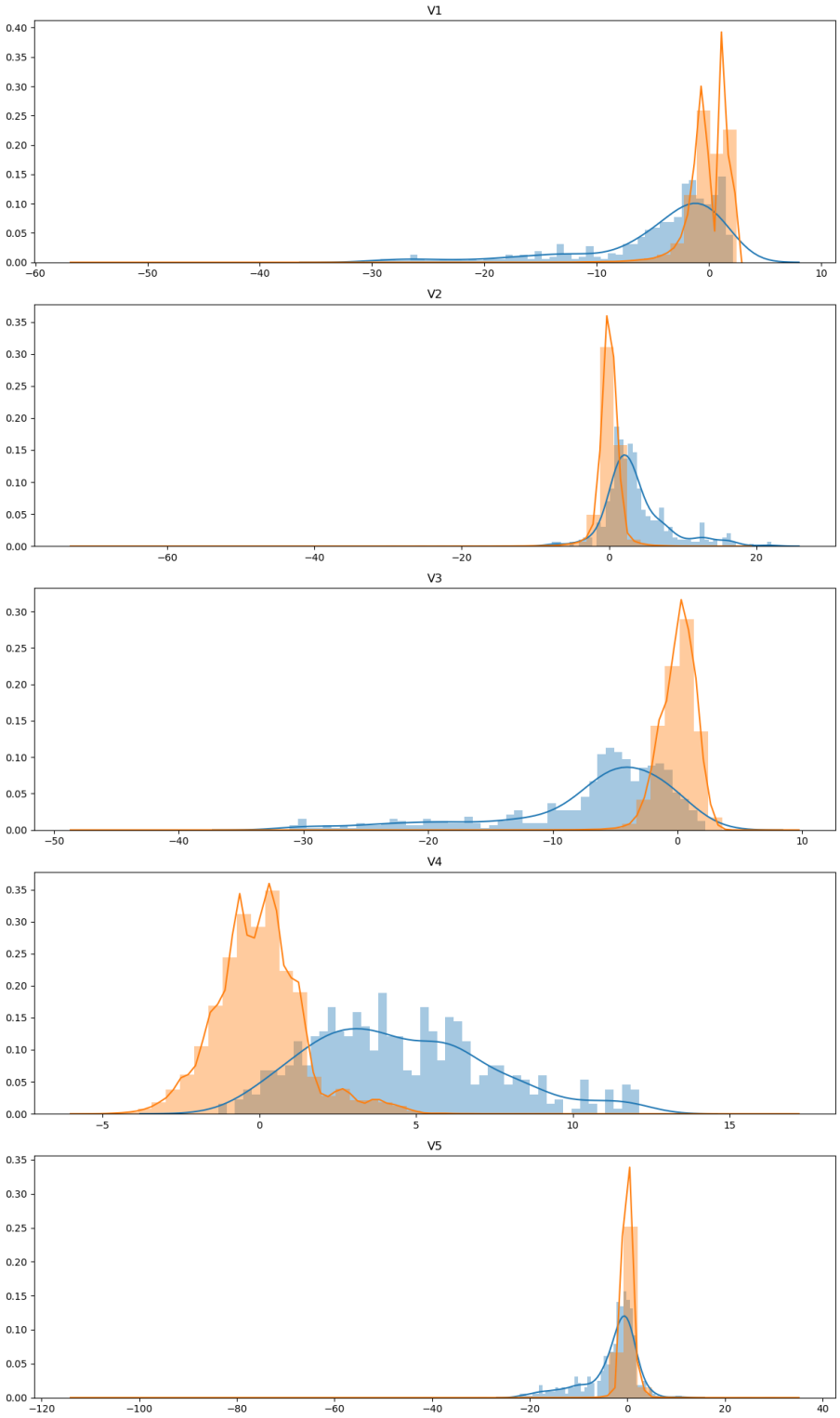

图 4: 结构化数据分布图 1-5

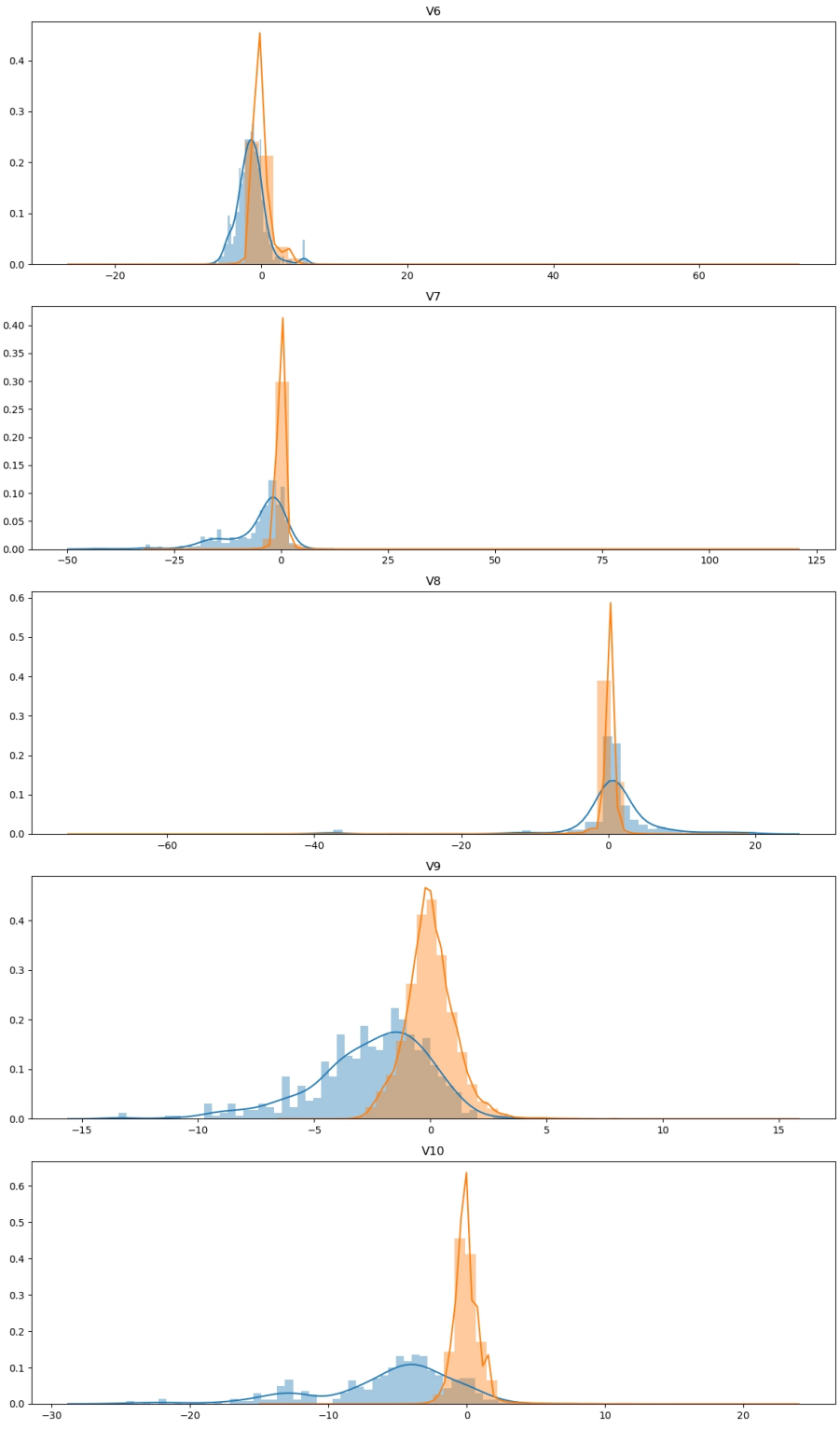

图 5: 结构化数据分布图 6-10

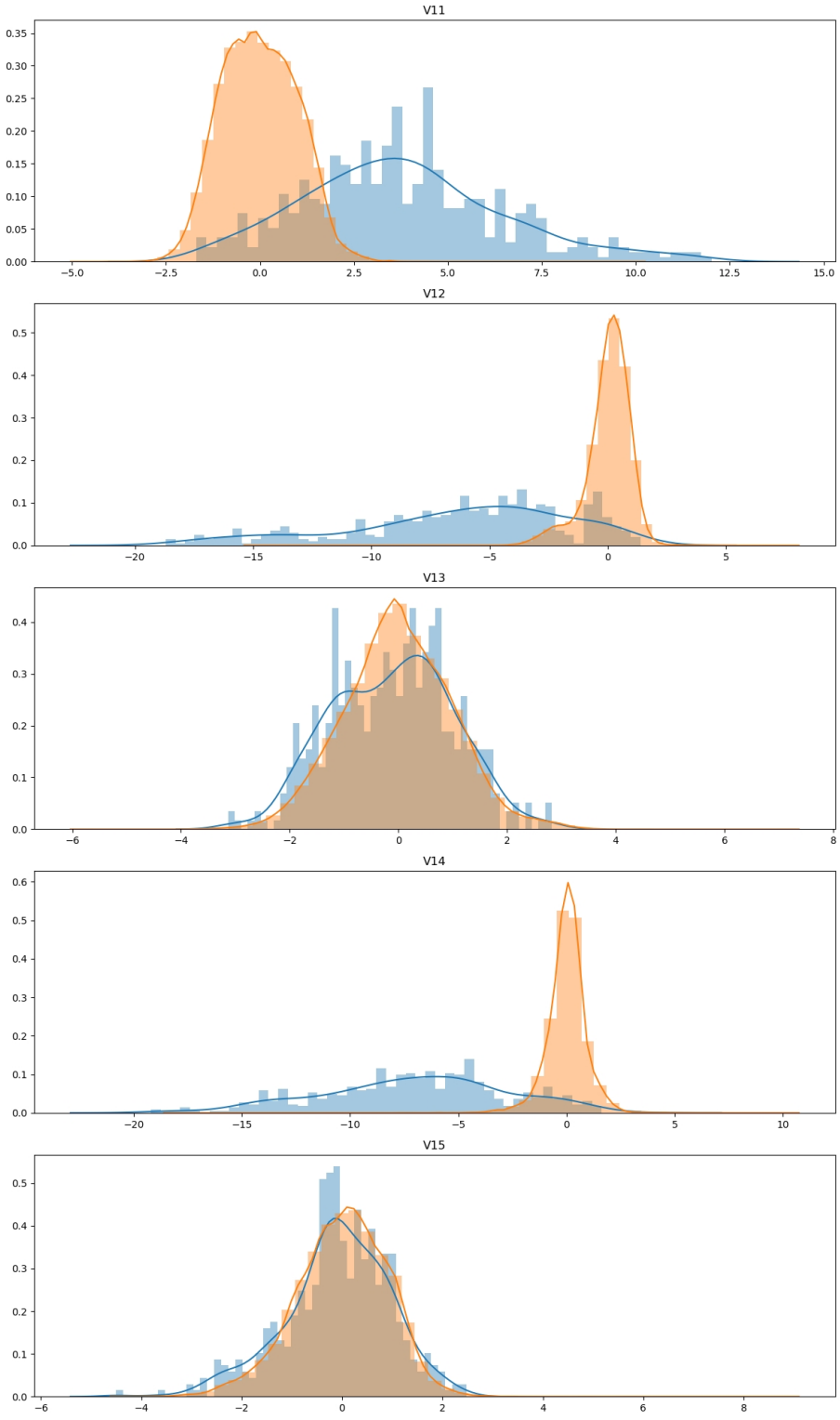

图 6: 结构化数据分布图 11-15

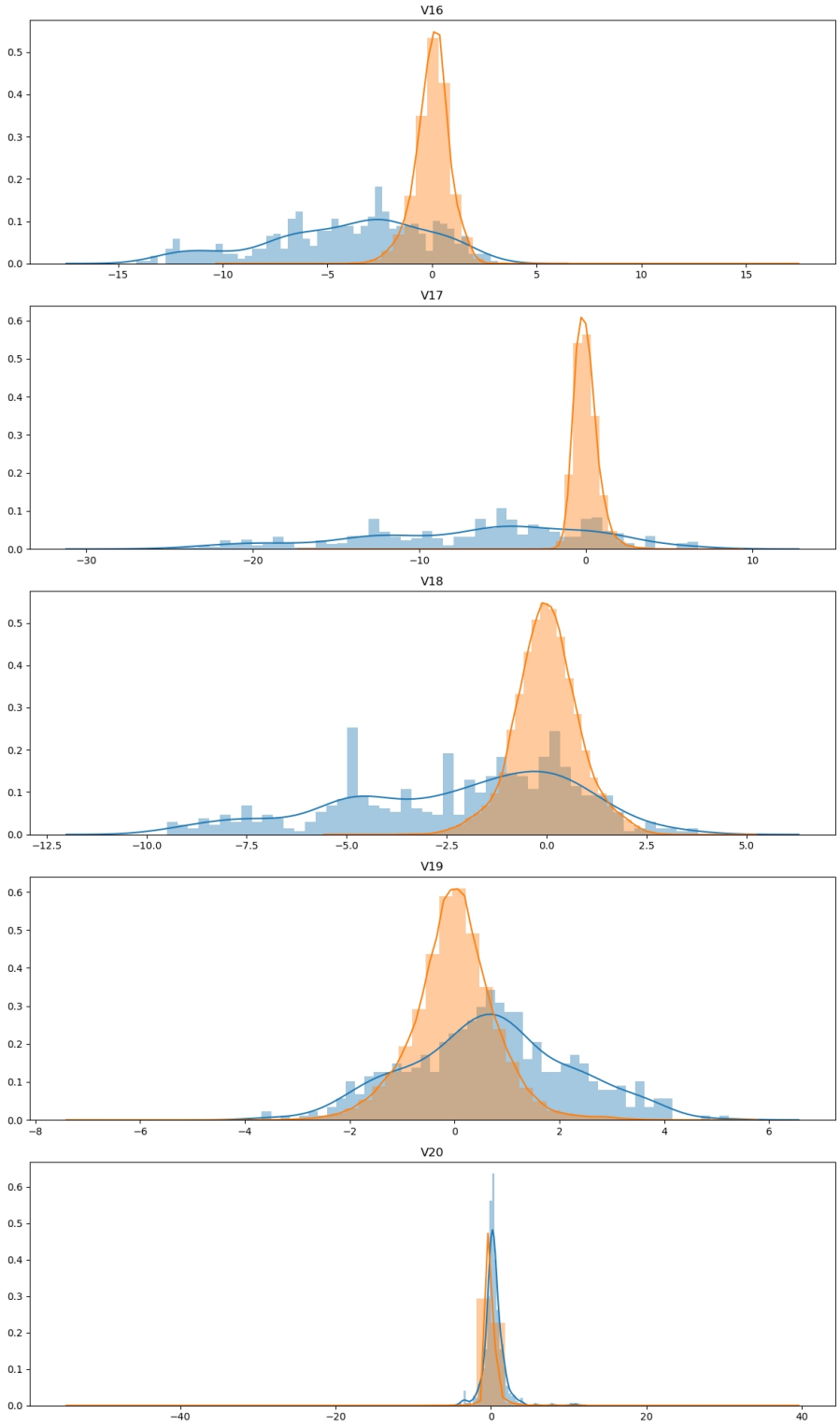

图 7: 结构化数据分布图 16-20

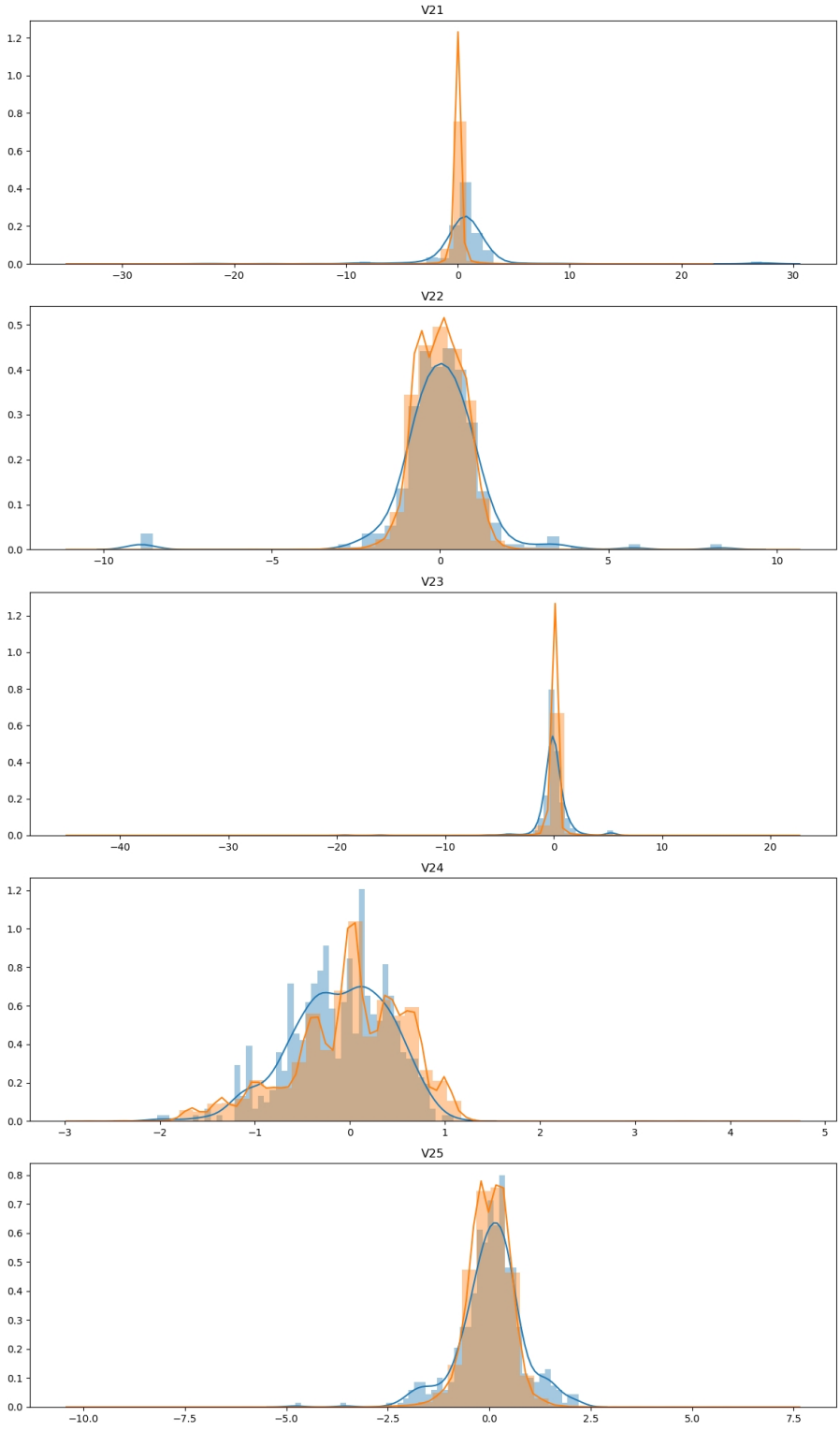

图 8: 结构化数据分布图 21-25

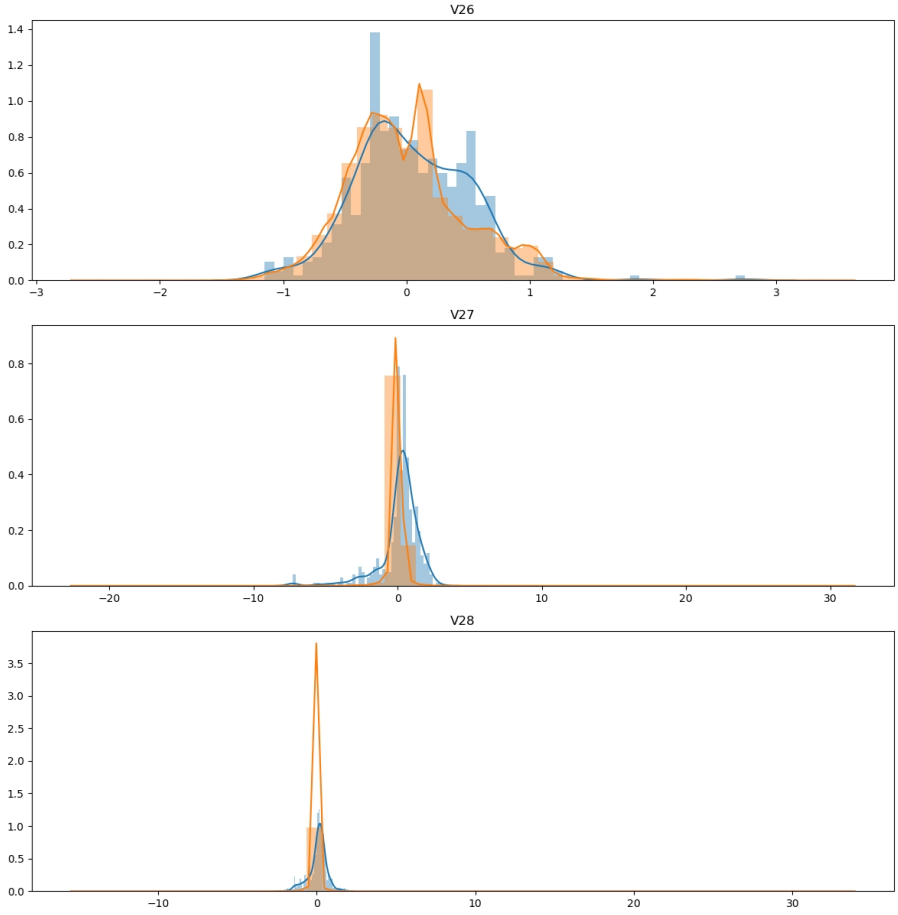

图 9: 结构化数据分布图 26-28

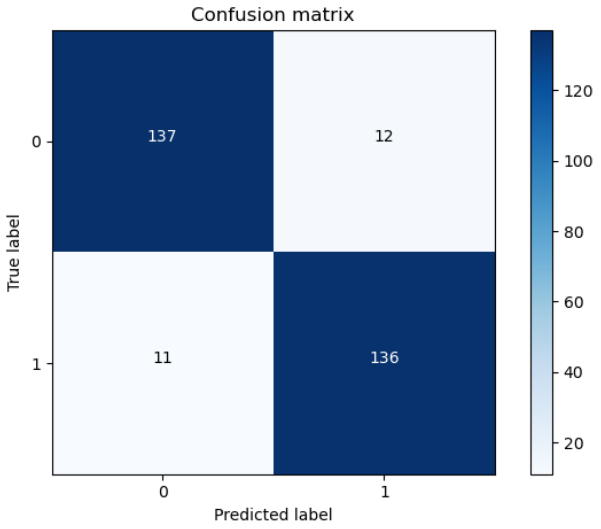

图 10: Confusion Matrix Under Sampling

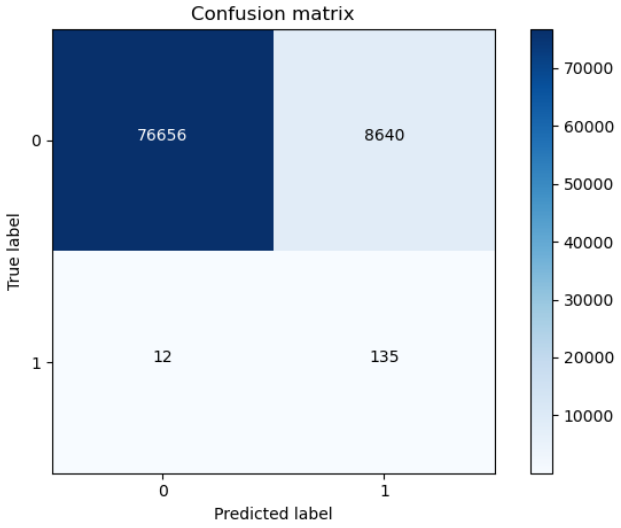

图 11: Confusion Matrix All

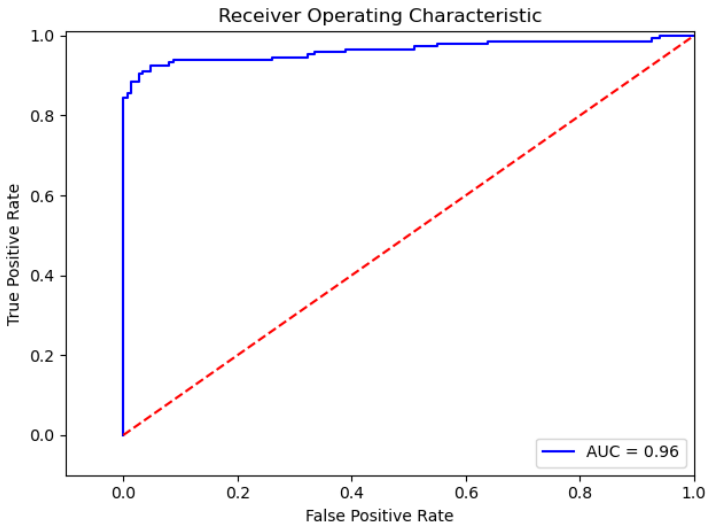

图 12: Confusion Matrix All

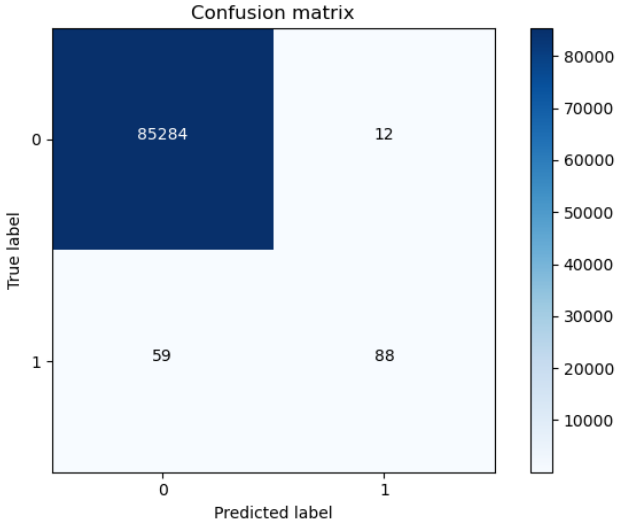

图 13: Confusion Matrix Raw Data

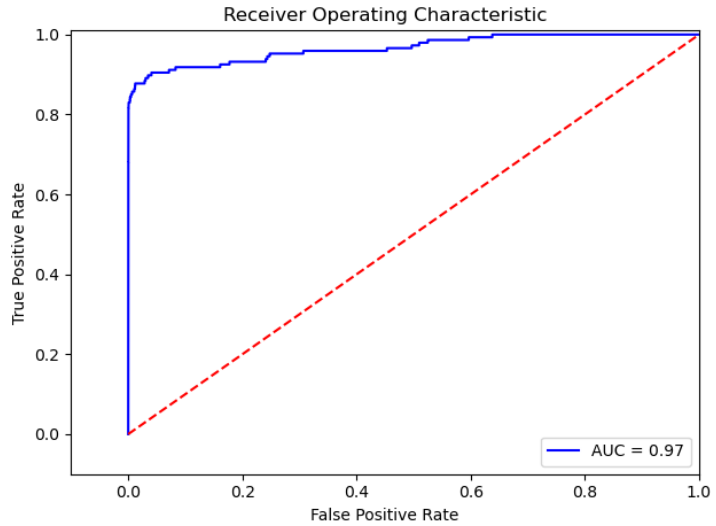

图 14: Confusion Matrix Raw Data

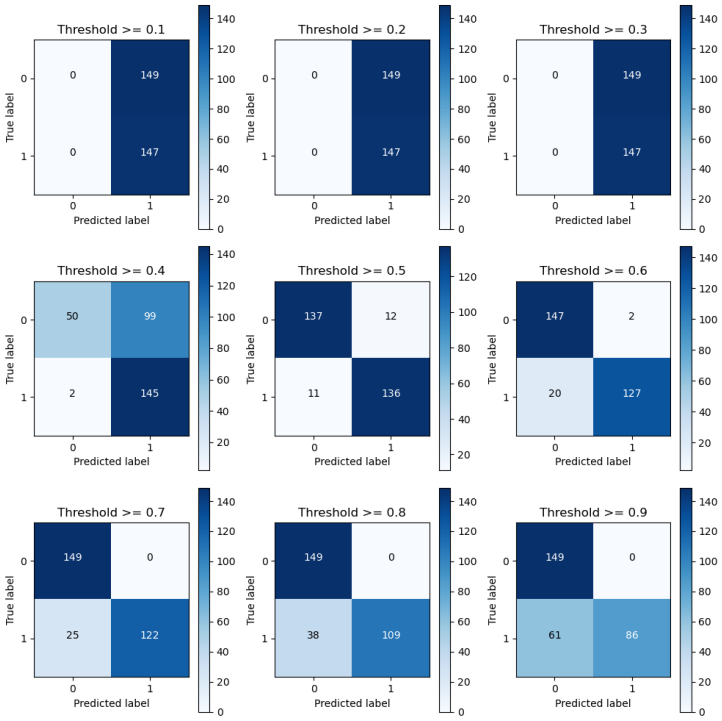

图 15: Confusion Matrix Probability