数据挖掘大作业Report

数据分析与预处理

我们的基础数据处理主要包括了对缺失值的填充和部分object类型数据的转换。

首先，我们筛选出了数值类型和object类型的数据，对于含有缺失值的数据，我们使用数据的均值进行了填充。

对于工作年限这样输入可能包含多种的数据，如：<1，>12等，我们进行了统一处理，转换为数字。同时，我们也把关键的class数据从类别转为了数字。

最后部分无法简单转换的类别数据，我们使用LabelEncoder对其进行了encode编码。

特别的，对于样本较多，噪音较大的train_internet数据，我们直接对含缺失值的行进行了舍弃。

算法实现与分析

三种算法及其效果

神经网络

我们选取 **MLP** 来实现该预测任务，使用 pytorch 构建模型，实现的模型结构如下：

```
class MLP(nn.Module):
    def __init__(self, num_features):
        super(MLP, self).__init__()
        self.layers = nn.Sequential(
            nn.Linear(num_features, 64),
            nn.ReLU(),
            nn.Linear(64, 64),
            nn.ReLU(),
            nn.Linear(64, 1),
            nn.Sigmoid()
        )
```

```
    def forward(self, x):
        return self.layers(x)
```

- **输入层**: 输入的特征数量由数据预处理部分确定，即 `num_features`。
- **隐藏层**: 网络包含两个隐藏层，每个隐藏层有 64 个神经元。这些隐藏层使用 ReLU 激活函数。
- **输出层**: 输出层只有一个神经元，并使用 Sigmoid 激活函数，将输出压缩到 0 和 1 之间。

损失函数和优化器

- **损失函数**: 使用了二元交叉熵损失（BCELoss），以适应概率输出的场景。
- **优化器**: 使用了 Adam 优化器，学习率设置为 0.001。

训练和验证过程

- **数据加载器**: 使用`DataLoader`进行批处理，训练集的批大小为 64，验证集亦同，但不进行数据打乱。
- **训练循环**: 训练共进行 100 个 epochs 。在每个 epoch 中，首先设置模型为训练模式，进行前向传播、损失计算、反向传播和参数更新。
- **验证循环**: 每个训练周期后，将模型设置为评估模式（`model.eval()`），在验证集上运行模型，不计算梯度（`torch.no_grad()`），计算整个验证集的平均损失和准确率。

验证集上的效果

经过 100 个 epoch 的训练后，模型在验证集上的 auc 为 **0.816115**

```
Epoch 100, Loss: 6.424751290978747e-07, Val Loss: 2.2966457773, Val AUC: 0.8161149225615878
```

随机森林

我们手动实现了随机森林的方法

算法实现：

决策树

决策树通过递归地分割数据集来构建树结构，每个节点选择最佳的特征和阈值进行分裂。算法的关键步骤如下：

1. **选择分裂点**：遍历所有特征和可能的阈值，计算信息增益，选择增益最大的分裂点。
2. **递归构建树**：根据选择的分裂点，将数据集分为左右子集，递归地在子集中继续分裂，直到达到最大深度或节点中的样本数少于最小分裂样本数。
3. **预测**：通过树结构遍历样本，从根节点开始，根据特征值和阈值向下递归，直至到达叶子节点，返回叶子节点的预测值。

随机森林

随机森林通过集成多个决策树来进行预测，可以增加模型的多样性。

随机森林通过集成多棵决策树来提高模型性能，每棵树都在一个随机样本上进行训练。算法的关键步骤如下：

1. **样本采样**：对原始数据集进行有放回的随机采样，生成多个与原数据集大小相同的子样本集。
2. **训练决策树**：在每个子样本集上训练一棵决策树。
3. **预测**：对每个样本，通过所有树进行预测，并通过多数投票或平均预测得到最终结果。

对于目标任务，我们设置的随机森林包含 15 棵深度最多为 15 的决策树，每个节点最少包含 2 个样本时才会分裂。

验证集上的效果

训练完毕后进行验证，随机森林在验证集上的 auc 为 **0.841708**

```
Training time: 950.6746928691864 seconds
Validation AUC: 0.8417084460276623
```

XgBoost

XGBoost 使用的是 CART 的决策树算法，它会将所有树的输出相加得到最终预测结果。XGBoost 在每一步添加新的树时，都会利用梯度下降算法来最小化损失函数。这个损失函数不仅衡量模型预测的准确性，还包括了正则化部分，并通过在构建树的过程中抽样特征列，提高了训练速度，减少过拟合。

我们使用的是 xgboost 库中的 XGBClassifier，超参数为：模型包括 100 棵决策树，学习率为 0.1，树的最大深度为 6。

在训练过程中，我们使用了交叉验证的方法，使用 `KFold` 进行交叉验证。在训练模型后，计算AUC作为性能指标。

训练完毕后进行验证，XgBoost 在验证集上的 auc 为 **0.89695**

```
Fold 1, Validation AUC: 0.9080480196216775
Fold 2, Validation AUC: 0.9053711051464581
Fold 3, Validation AUC: 0.8919678791528621
Fold 4, Validation AUC: 0.8891583982990786
Fold 5, Validation AUC: 0.8902101090883582
auc=89.695.csv
```

不同算法的差异

MLP

MLP 适合捕捉数据中的非线性复杂模式，特别是在大规模数据集上。优点是其具有强大的表示能力，尤其在处理复杂模式和高维数据时表现出色，并且可以通过增加层数和神经元数灵活调整模型复杂度。但缺点是需要大量的数据来有效训练，避免过拟合。并且训练时间长，参数多，调参复杂。

随机森林

随机森林是一种集成学习方法，通过构建多个决策树并将它们的预测结果进行平均或多数投票来提高预测准确性。每棵树的构建基于随机选择的样本和特征，增加了模型的多样性。优点是训练速度快，容易并行化。对于不平衡数据集表现良好，具有很好的抗噪声能力。并且不需要太多的参数调整即可达到不错的性能。缺点是在某些噪音较大的复杂回归任务中表现不如梯度提升树。在某些高维数据上可能会表现不佳。

XGBoost

XGBoost 是一种优化的分布式梯度提升库，通过逐步构建加性模型来优化任意可微分的损失函数。并且提供了正则化以避免过拟合。优点是训练速度快，效率高，可以处理大规模数据，尤其是在结构化数据。缺点是需要仔细调整参数（如树的数量、深度、学习率等）。相比随机森林，可能更容易过拟合，尤其是在数据量较少时。

在本次大作业的任务中，XGBoost 的表现最为出色。

特征工程 &不同特征的影响

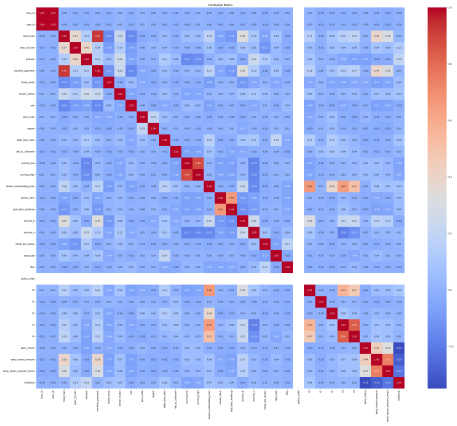
基于随机森林的打伪标签方法

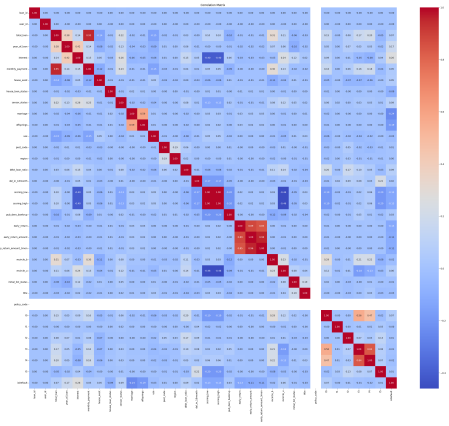
该处理的出发点，是我们观察发现，train_internet数据相比普通的train数据，多了一个sub_class项。我们认为这样更细分类的sub_class可能有利于数据的特征分析，因此尝试为train_public数据同样增加这样一个sub_class标签。我们选用的方法，是利用train_internet数据和train_public数据的相似性，把train_internet和train_public数据都按大类划分。对于每个大类，利用train_internet数据训练一个基于随机森林算法的分类器，去完成sub_class 的分类。然后再利用该分类器去给train_public数据打伪标签。我们发现这样打出的伪标签很好的提升了最后分类的效果。

交互项 & 二阶项构造

本节尝试引入**交叉特征（Cross Features）**，帮助模型捕捉到不同特征之间的关联和相互作用，从而使其能够更准确地建模复杂的数据关系。

我们通过对 public 数据集和 internet 数据集绘制相关关系图（左为public数据，右为internet数据），可以发现两组数据中 early_return_amount , early_return 和 early_return_amount_3mon 存在一定的线性相关关系，这种关系在internet数据中尤为明显。结合他们的实际含义不难解释这种现象：**对于大多数人来说，还款次数和还款总量之间存在一定的线性关系，其"斜率"可以理解为该对象每次还款的期望值。**而public数据集中这种线性关系相比 internet数据更弱，这恰恰符合我们对交互项数据分布的期待：**public数据中每个数据点的"单次还款期望"并不相同，而这一特征相比还款总量排除了还款次数带来的噪音影响，很有可能对模型的学习有所帮助。**





进一步，通过以 early_return 为因子做方差分析可以看到，不同 early_return 水平下的还款总额存在明显的差异，这说明在public数据集中 early_return 和

early_return_amount（3mon）之间仍存在一定的关系（只不过这种关系未必是线性的）

	F_value	P_value
isDefault	374.002993	0.000000e+00
early_return_amount	608.753635	0.000000e+00
year_of_loan	433.435465	0.000000e+00
early_return_amount_3mon	288.342142	4.294489e-289

基于上述分析，我们把二者相除所得到的值作为特征引入训练集。在验证集和测试的最终结果如下

	Validation	Test
Baseline(不引入额外交叉项)	90.032	88.616
+early_return_amount_3mon_early_return	89.675	90.307
+early_return_amount_early_return	89.564	90.541
+early_return_amount_3mon_early_return & early_return_amount_early_return	89.614	**90.776**

可以看到，尽管在验证集上模型性能并没有明显提升，但在测试集上性能整整提升了2个点，这说明引入的交互项特征可以大幅提高模型的鲁棒性，使其在测试集上有更好的性能。而另一个有趣的发现是，`early_return_amount_3mon_early_return` 和 `early_return_amount_early_return` 两个特征只需要引入一个就可以令模型性能大幅提升，而两个特征同时引入只有轻微的增加，这说明这两个特征可能有极大的相关性，模型学习时只需要学习其中一个特征即可实现较大的性能提升。

分工情况

- 苏禹畅：实现 XGBoost 算法，进行特征工程
- 周韧平：完成数据分析与预处理，实现 MLP
- 张航宇：手动实现随机森林算法，进行测试