

数据集成第三次作业

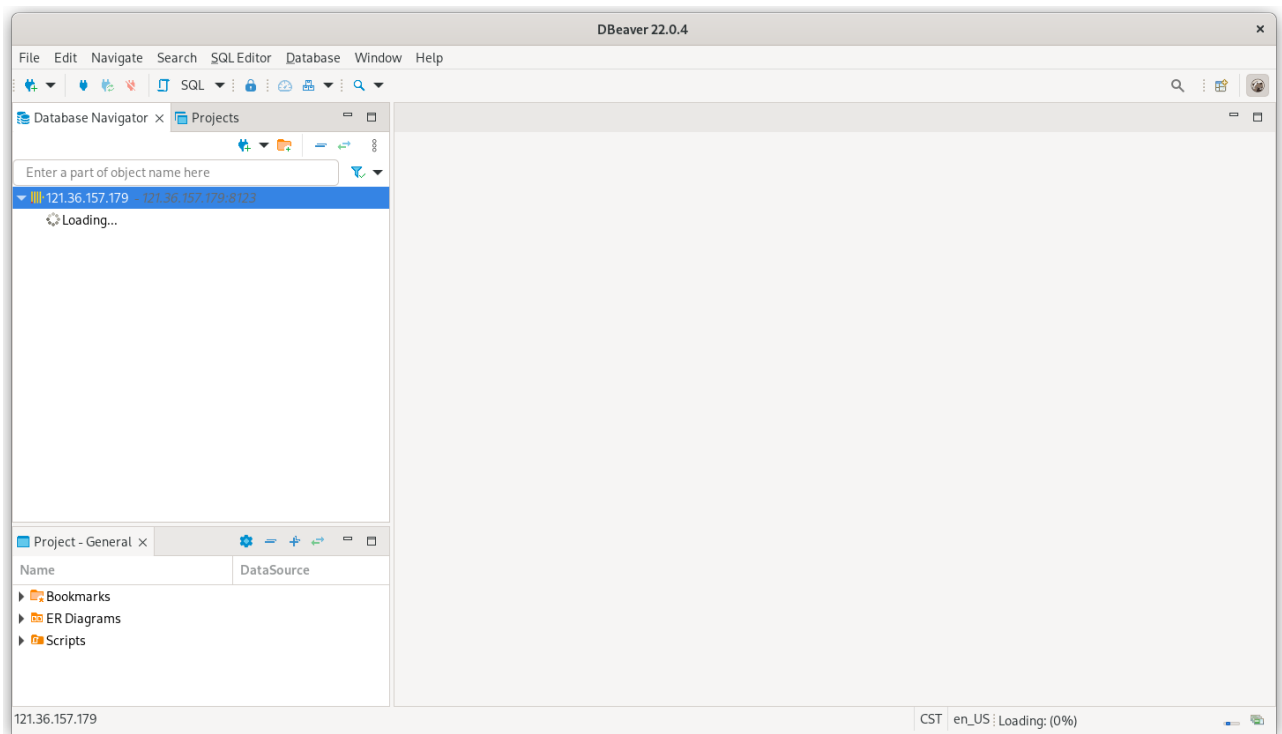
小组成员：黄相淇、王子悦、张刘洋、周辰熙

选择主题：主题一：客户星级和信用等级评估，利用机器学习的方法对用户**星级**和**信用等级**进行评估

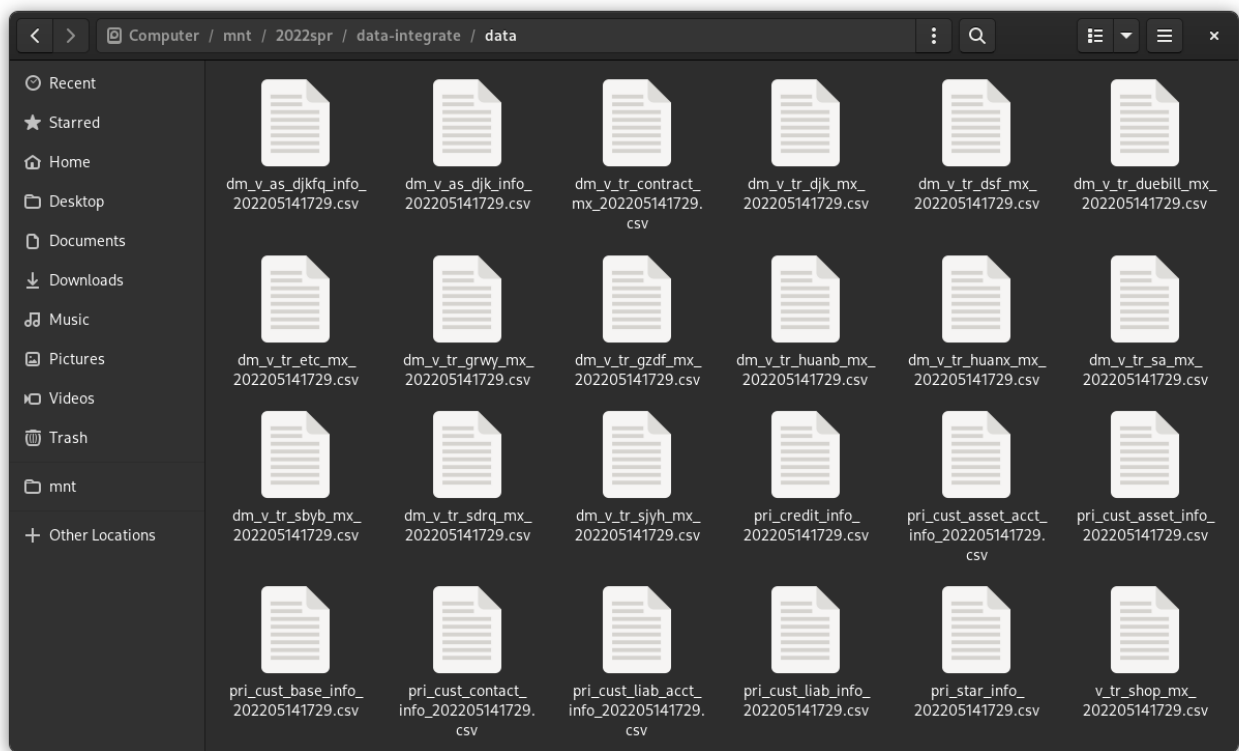
代码地址 <https://github.com/ChenxiZhou0619/Data-Integration-2022Spr>

1. 数据准备

在第二次作业的基础上，本次作业我们选用了 **DBeaver** 工具将位于服务器上的clickhouse中的数据下载至本地，以便于后续读取、处理



数据以csv文件格式进行存储



2.数据读取和预处理

2.1 读取

首先使用pandas读取csv文件

e.g. `pri_cust_asset_info`表

```
cust_asset = pandas.read_csv(  
    "/mnt/2022spr/data-  
integrate/data/pri_cust_asset_info_202205141729.csv",  
    usecols=[  
        "uid",          # 证件号码, 用作连接各个pandas dataframe  
        "all_bal",      # 总余额  
        "avg_mth",      # 月日均  
        "avg_year",     # 年日均  
        "sa_bal",       # 活期余额  
        "td_bal",       # 定期余额  
        "fin_bal",      # 理财余额  
        "sa_crd_bal",   # 卡活期余额  
        "sa_td_bal"     # 定活两便  
    ]  
)
```

由于表中部分数据列的值均相同 (e.g. etl日期、某些列均为0)，因此我们只读取部分行

e.g. `pri_star_info`表

```
pri_star_table = pandas.read_csv (
    "/mnt/2022spr/data-
    integrate/data/pri_star_info_202205141729.csv"
)
```

该表只有两列，分别为uid和star_level，标明了数据的标签

2.2 表连接

pandas也提供了类似于数据库的join操作

e.g. 连接上面读取的两个表

```
res = pandas.merge(pri_star_table, cust_asset, how='left',
    on='uid')
res = res[(res['star_level']!=-1)]
res = res[(res['all_bal'].notnull())]
```

`pandas.merge` 操作将两个pandas dataframe进行连接，上面的代码标明是left join，以uid进行连接。我们先进行模型训练，因此将合并后标签为-1的行过滤掉；同时由于连接，也有可能部分行存在空值的情况，也将其过滤掉

2.3 数据预处理

上述操作得到的依旧是一个pandas dataframe,无法直接进行模型训练，因此需要将其转为可以使用模型训练的数据结构 (e.g. numpy.ndarray / torch.tensor)，同时进行数据预处理 (e.g. 中心化、归一化)

```
train_mapper = DataFrameMapper ([
    (['all_bal'], sklearn.preprocessing.StandardScaler()),
    (['avg_mth'], sklearn.preprocessing.StandardScaler()),
    (['avg_year'], sklearn.preprocessing.StandardScaler()),
    (['sa_bal'], sklearn.preprocessing.StandardScaler()),
    (['td_bal'], sklearn.preprocessing.StandardScaler()),
    (['fin_bal'], sklearn.preprocessing.StandardScaler()),
```

```

        (['sa_crd_bal'],
 sklearn.preprocessing.StandardScaler()),
        (['acct_bal'], sklearn.preprocessing.StandardScaler()),
    ])

    X = np.round (train_mapper.fit_transform(res.copy()), 2)    #
    得到向量化数据

    labels_mapper = DataFrameMapper([
        (['star_level'], None)
    ])

    Y = np.round (labels_mapper.fit_transform(res.copy()))    #
    得到数据标签

```

上述代码定义了一个mapper,将data frame的每一列数据进行映射,
`sklearn.preprocessing.StandardScaler()` 将数据进行标准化,以便于模型训练。之后`np.round`将其转化为可供训练使用的numpy数组,保留两位小数

3. 使用机器学习模型进行训练

首先将数据划分为训练集和测试集

```

X_train, X_test, Y_train, Y_test = train_test_split (
    X, Y, test_size=0.2
)    # 以 8 : 2 的比例划分训练集和测试集

```

3.1 决策树

混淆矩阵:

```
[[17002    982     21      2      0      0      0      0      0]
 [   853   6991    246      4      2      0      0      0      0]
 [    31    365   6708    354     24      0      0      0      0]
 [    10      7    346   1598    247     17      1      0      0]
 [     6      2     18    272   1365    121      0      0      0]
 [     4      0      1      5     84    774      3      0      0]
 [     0      0      0      0      1      1     30      0      0]
 [     0      0      0      0      0      0      0      0      4]
 [     0      0      0      0      0      0      0      0      2]]
```

准确度: 0.8939

3.2 支持向量机

混淆矩阵:

```
[[17151    606     52      2      2      1      0      0      0]
 [  2574   5141    354      3      3      3      0      0      0]
 [    35    657   6400    480     34      3      0      0      0]
 [     9      2    757   1023    411     13      0      0      0]
 [    13      0      2    372   1205    243      0      0      0]
 [     2      0      0      0    155    766      0      0      0]
 [     0      0      0      0      0      6     16      0      0]
 [     0      0      0      0      0      2      0      3      0]
 [     0      0      0      0      0      0      0      0      3]]
```

准确度: 0.8223

由于数据量较大的问题（用于评估star的数据超过1w条，用于评估credit的数据超过19w条），因此在后续的实验结果展示中我们将不选择支持向量机作为模型

3.3 神经网络

混淆矩阵:

```

[[16441  1566    55     0     3     0     0     0     0]
 [ 1244  6485   392     1     0     0     0     0     0]
 [   29   499  6346   500    19     0     0     0     0]
 [   14     4   696  1110   419     9     1     0     0]
 [   16     6     2   332  1208   178     1     0     0]
 [    3     1     0     2   212   669     0     4     1]
 [    0     0     0     0     0    20     8     1     0]
 [    0     0     0     0     1     1     0     4     0]
 [    0     0     0     0     0     0     0     0     1]]

```

准确度: 0.8381

神经网络我们使用了**pytorch**作为框架，其中网络的具体信息如下

1. 网络结构

```

class MyNetwork(nn.Module):
    def __init__(self):
        super(MyNetwork, self).__init__()
        self.linear_relu_stack = nn.Sequential (
            nn.Linear(8, 20),
            nn.SELU(),
            nn.Linear(20, 10),
            nn.SELU(),
        )
    def forward(self, x):
        logits = self.linear_relu_stack(x)
        return logits

```

网络部分，数据的输入维度为8，中间有一个维度为20的隐藏层，输出维度为10，其中最大的维度表明网络的预测结果，激活函数选择**SELU**

2. 训练参数

学习率设定为0.001，优化算法选择Adam算法，损失函数选择交叉熵损失函数，迭代次数选择为10

```
loss_fn = nn.CrossEntropyLoss()
learning_rate = 1e-3
optimizer = torch.optim.Adam(model.parameters(),
                               lr=learning_rate)

epochs = 10
    for t in range(epochs):
        print(f"Epoch {t+1}\n-----")
        train_loop(train_data_loader, model, loss_fn,
                    optimizer)
        test_loop(test_data_loader, model, loss_fn)
    print("Done!")
```

运行截图:

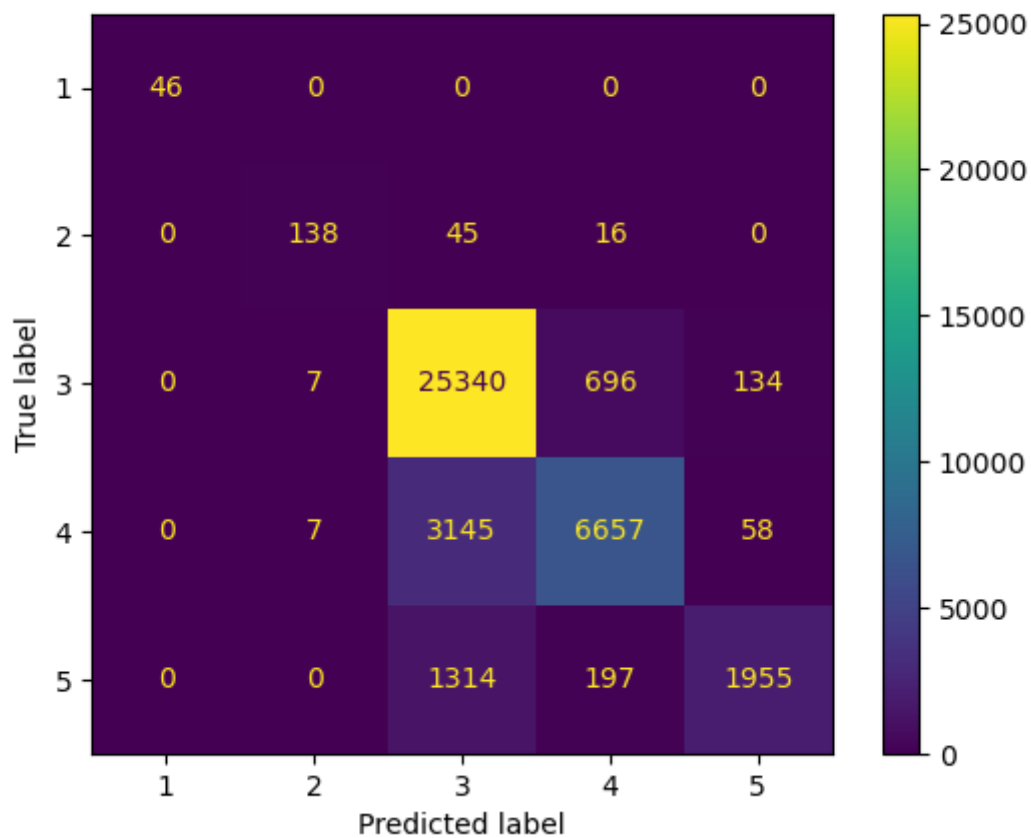
```
Epoch 10
-----
loss: 0.404194 [ 0/154012]
loss: 0.680762 [ 6400/154012]
loss: 0.415925 [12800/154012]
loss: 0.319984 [19200/154012]
loss: 0.350248 [25600/154012]
loss: 0.326535 [32000/154012]
loss: 0.502849 [38400/154012]
loss: 0.373350 [44800/154012]
loss: 0.203996 [51200/154012]
loss: 0.331480 [57600/154012]
loss: 0.297658 [64000/154012]
loss: 0.345409 [70400/154012]
loss: 0.425146 [76800/154012]
loss: 0.331698 [83200/154012]
loss: 0.522846 [89600/154012]
loss: 0.285024 [96000/154012]
loss: 0.278727 [102400/154012]
loss: 0.356062 [108800/154012]
loss: 0.312718 [115200/154012]
loss: 0.387348 [121600/154012]
loss: 0.346462 [128000/154012]
loss: 0.345772 [134400/154012]
loss: 0.372468 [140800/154012]
loss: 0.323735 [147200/154012]
loss: 0.377576 [153600/154012]
Test Error:
  Accuracy: 83.8%, Avg loss: 0.385728
```

以上部分，我们以用户star的评估作为示例展示实现过程，credit的评估过程类似

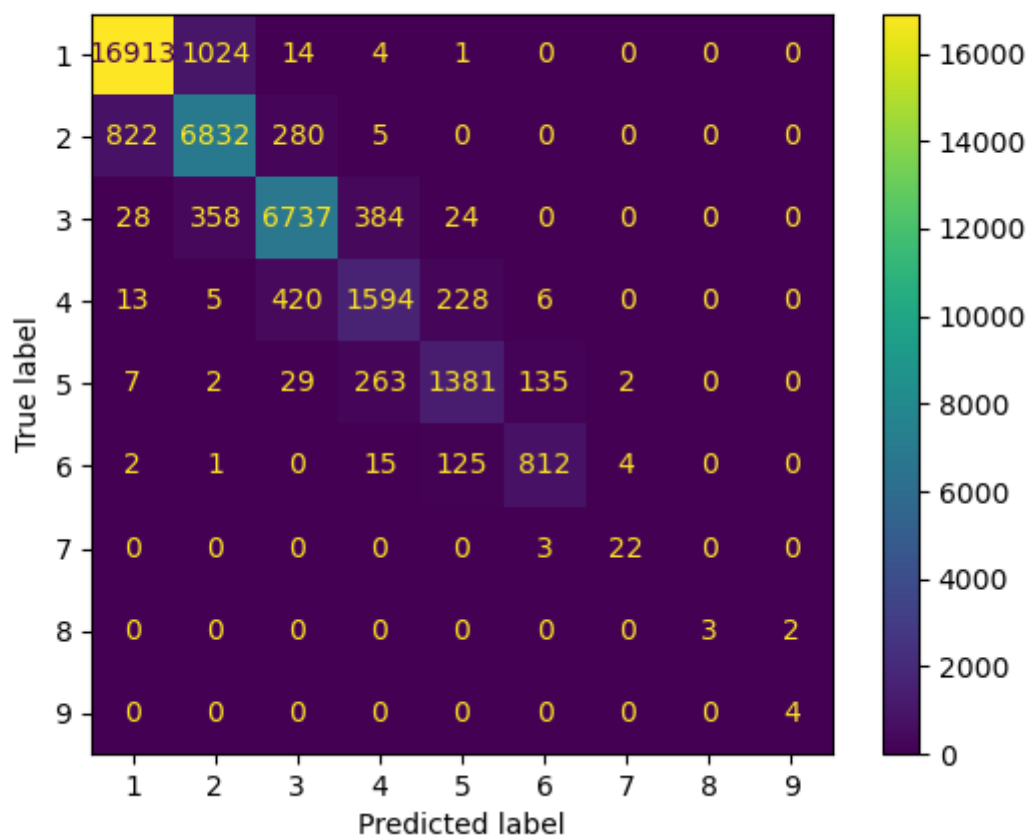
4. 实验结果展示

4.1 决策树

信用 **credit**评价：准确率 85.97%，下为混淆矩阵

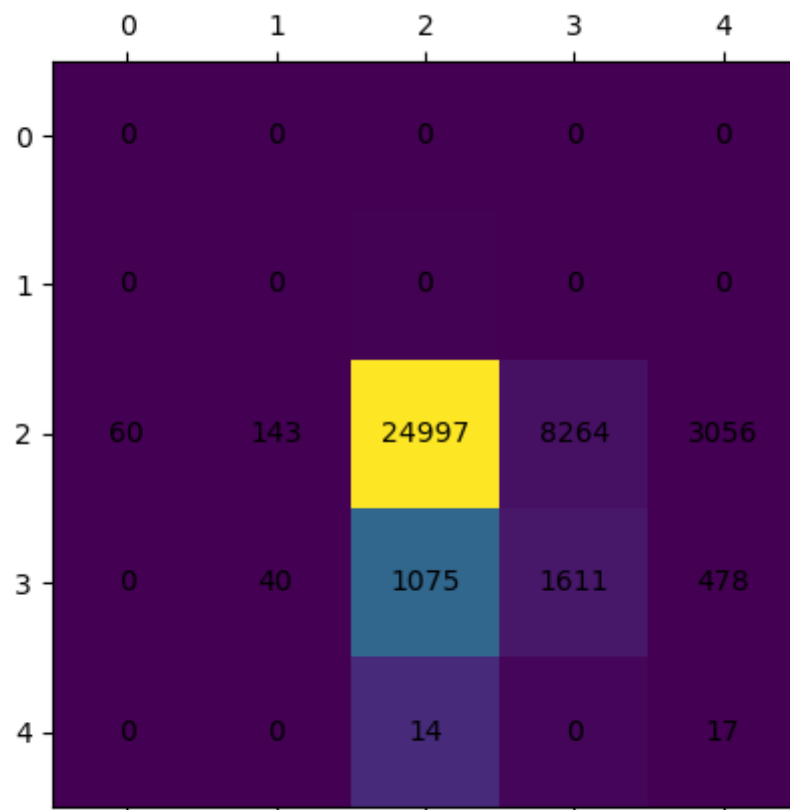


星级 star评价：准确率 89.07%，下为混淆矩阵



4.2 神经网络

信用 credit评价：准确率仅有66.97%，下为混淆矩阵，效果不好



星级 star评价：准确率 83.67%，下为混淆矩阵

