AI22 PROJECT TASK3

FINAL

1. Introduction

本实验的任务是利用机器学习的方式，进行直播平台恶意退款的识别。

根据给定的数据集（交易的基本信息+是否退款的标签），采用 RFM 框架，构建特征工程，并用神经网络和 self-sttention 自注意力机制对测试数据集进行训练，根据输入交易的特征预测其是否会发生退款。

2. Dataset

虎牙平台交易数据集，利用苹果渠道政策（120 天内无条件退款），虎牙平台中的恶意退款行为，17 列，730035 行，正常交易 72743 笔，恶意退款 2582 笔。

字段	信息	备注
session_id	订单会话 id	字段格式为：用户唯一识别 id/用户交易发生的 IP 地址
version	版本	苹果充值程序的版本，而不是 app 的版本
prod_id	产品 ID	用户通过苹果充值实际购买物品的名称（虎牙币、金豆、银豆、开通会员、守护、续费贵族）
prod_name	产品名称	虎牙币、金豆、银豆、贵族、守护
amount	金额	实际花费的金钱
unit	币种	苹果的实际结算币种
exchange_rate	汇率	不同币种对应的汇率
rmb_amount	以人民币结算的金额	用户实际用币种支付的金额
invoice_amount	开票金额	可以开发票的金额，比例是固定的 70%
ch_fee	渠道费用	苹果收取的渠道费，ios 是 30%的费用
ch_fee_rate	比例	在苹果渠道下，虎牙可以收到的费用比例
status_code	订单状态	包含三种，其中 CODE_REAR_RISK_FAIL 属于前置拦截
ch_deal_time	渠道处理时间	订单交易的时间，订单完成状态，订单状态显示的时间
refund_time	退款时间	无退款则为空
refund_desc	退款描述	0 是用户发起，1 是平台发起用户发起指的是用户自己提交退款申请，平台发起指的是 apple 接到它的上游渠道,例如银行或者信用卡渠道的申请进行的退款
refund_status_code	退款状态码	有状态码就是有退款

3. Proposed method

3.1 feature engineering

RFM 框架：

recency - 最近购物的时间

frequency - 购物频率

monetory - 消费金额

周期：
三天、七天、十五天、一个月、三个月
交易特征：
用户过去购买商品的次数和
用户过去购买金额的总和、均值、最大值、标准差
用户过去交易的当天购买时长均值、最大值、标准差

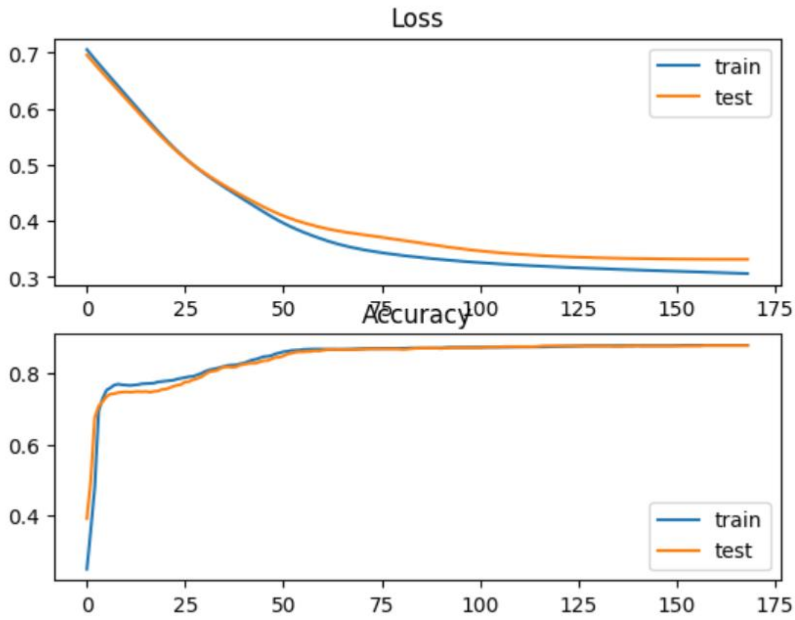
采用交易聚合策略，最终构建的特征为：
周期*交易特征　　一共 40 维

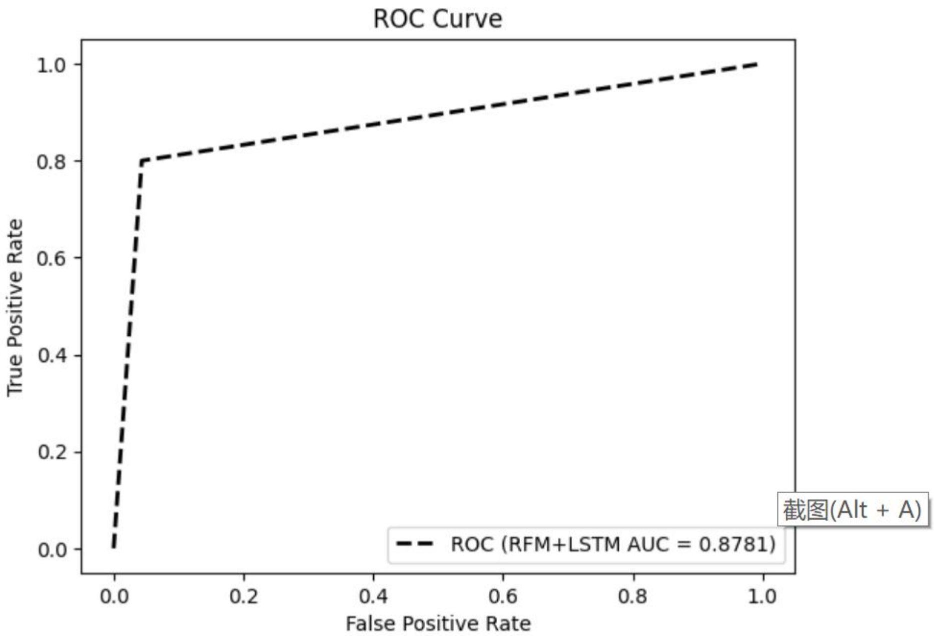
StandardScaler：数据标准化，针对每一个特征维度来做的，而不是针对样本。
划分训练集和数据集
under_sampling：欠采样，即去除一些反例使得正、反例数目接近。原数据样本中正负样本不均衡，所以采用随机欠采样

3.2　Model

1.LSTM：长短期记忆神经网络

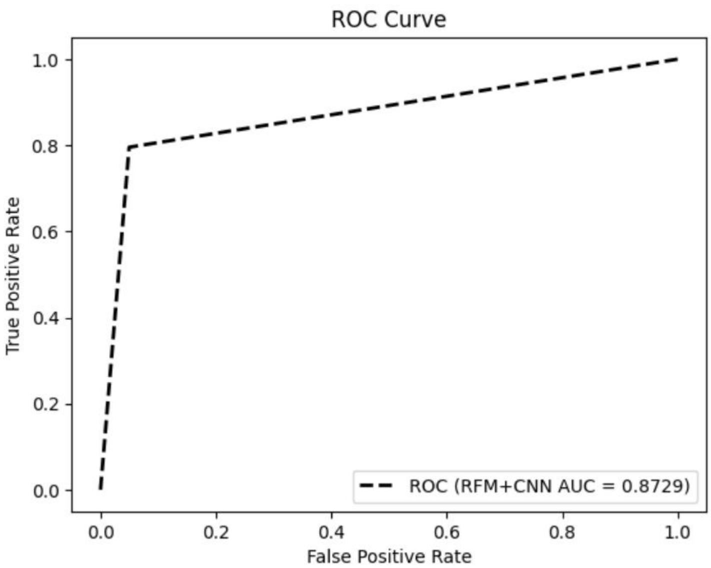
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 1, 40)]           0

 lstm (LSTM)                 (None, 1, 50)             18200

 lstm_1 (LSTM)               (None, 50)                20200

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 38,451
Trainable params: 38,451
Non-trainable params: 0
_________________________________________________________________
```

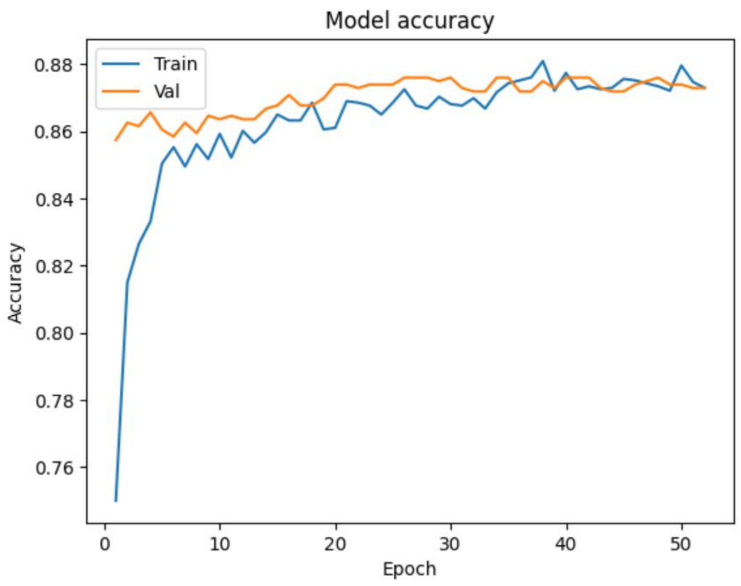

Loss
train
test
Accuracy
train
test

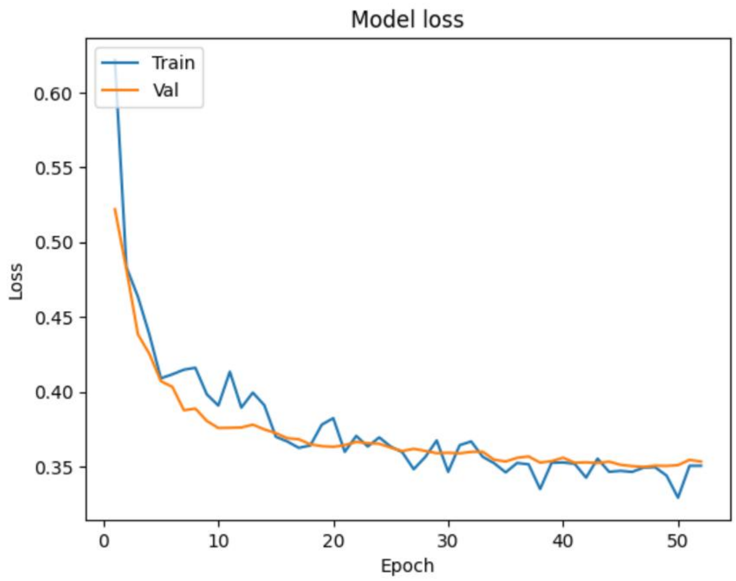

ROC Curve
True Positive Rate
False Positive Rate
ROC (RFM+LSTM AUC = 0.8781)
截图(Alt + A)

2.CNN：卷积神经网络

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_2 (Conv1D)           (None, 39, 32)            96

 batch_normalization_2 (Batc  (None, 39, 32)           128
 hNormalization)

 dropout_3 (Dropout)         (None, 39, 32)            0

 conv1d_3 (Conv1D)           (None, 38, 64)            4160

 batch_normalization_3 (Batc  (None, 38, 64)           256
 hNormalization)

 dropout_4 (Dropout)         (None, 38, 64)            0

 flatten_1 (Flatten)         (None, 2432)              0

 dense_3 (Dense)             (None, 64)                155712

 dropout_5 (Dropout)         (None, 64)                0

 dense_4 (Dense)             (None, 1)                 65

=================================================================
Total params: 160,417
Trainable params: 160,225
Non-trainable params: 192
_________________________________________________________________
```







3.逻辑回归模型

4.GaussianNB 模型

5.svm 模型

6.KNN 模型

7.LDA 模型

8.随机森林模型

9.LSTM+attention:

```python
class Mylstm_attn(nn.Module):
    def __init__(self, batch):
        super().__init__()
        self.hidden = 128
        self.batch = batch
        self.attn_weight = nn.Parameter(torch.randn(self.batch, 1, self.hidden))
        self.rnn = nn.LSTM(input_size=1, hidden_size=self.hidden // 2, batch_first=True, num_layers=1,
                           bidirectional=True)
        self.project = nn.Linear(self.hidden, 2)
        self.activation = nn.Sigmoid()
        self.attn_drop = nn.Dropout(p=0.3)
        self.lstm_drop = nn.Dropout(p=0.3)

    def attention(self, H):
        M = torch.tanh(H)
        a = torch.softmax(torch.bmm(self.attn_weight[:M.shape[0]], M), 2)
        a = torch.transpose(a, 1, 2)
        return torch.bmm(H, a)

    def forward(self, X):
        output = X.unsqueeze(2)
        output, _ = self.rnn(output)
        output = self.lstm_drop(output)
        output = output.transpose(1, 2)
        output = self.attention(output)
        output = output.transpose(1, 2)
        output = self.attn_drop(output)
        output = self.project(output)
        return output.squeeze()

    def test(self, X):
        output = X.unsqueeze(2)
        output, _ = self.rnn(output)
        output = output.transpose(1, 2)
        output = self.attention(output)
        output = output.transpose(1, 2)
        output = self.project(output)
        output = output.squeeze()
        output = self.activation(output)
        output = torch.argmax(output, dim=1)
        return output
```

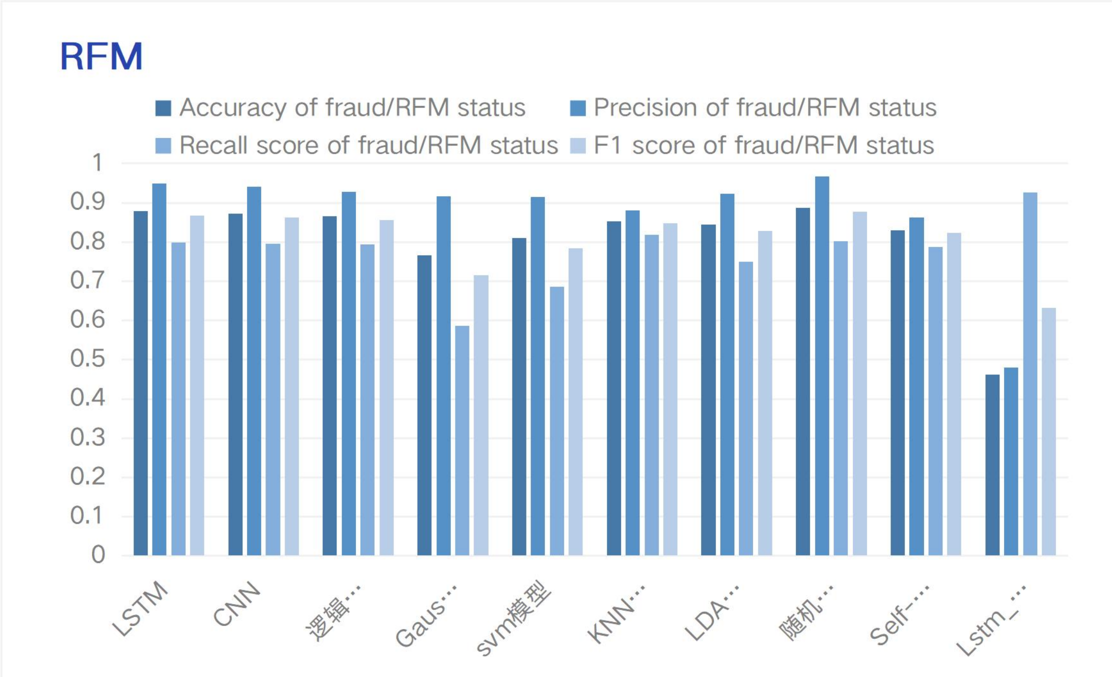
```python
        else:
            attention_score = self.mul_attention_score(Q, V, head)
            V = V.reshape(V.shape[0], V.shape[1], head, -1).transpose(1, 2)
            V = V.reshape(V.shape[0] * head, V.shape[2], V.shape[3])
            result = torch.bmm(attention_score, V)
            result = result.transpose(1, 2)
            result = result.reshape(int(result.shape[0] / head), head, result.shape[1], result.shape[2])
            result = result.reshape(result.shape[0], head * result.shape[2], result.shape[3]).transpose(1, 2)
            V=V.transpose(1,2)
            V=V.reshape(int(V.shape[0]/head),head,V.shape[1],V.shape[2])
            V=V.reshape(V.shape[0],head*V.shape[2],V.shape[3]).transpose(1,2)
        result=result+V
        result=self.ffn_add_norm(result)
        result=result.transpose(1,2)
        result=self.A(result)
        # result
        return result
```

10. Self-attention:

```python
class Mylstm_attn(nn.Module):
    def __init__(self, batch):
        super().__init__()
        self.hidden = 128
        self.batch = batch
        self.attn_weight = nn.Parameter(torch.randn(self.batch, 1, self.hidden))
        self.rnn = nn.LSTM(input_size=1, hidden_size=self.hidden // 2, batch_first=True, num_layers=1,
                           bidirectional=True)
        self.project = nn.Linear(self.hidden, 2)
        self.activation = nn.Sigmoid()
        self.attn_drop = nn.Dropout(p=0.3)
        self.lstm_drop = nn.Dropout(p=0.3)

    def attention(self, H):
        M = torch.tanh(H)
        a = torch.softmax(torch.bmm(self.attn_weight[:M.shape[0]], M), 2)
        a = torch.transpose(a, 1, 2)
        return torch.bmm(H, a)

    def forward(self, X):
        output = X.unsqueeze(2)
        output, _ = self.rnn(output)
        output = self.lstm_drop(output)
        output = output.transpose(1, 2)
        output = self.attention(output)
        output = output.transpose(1, 2)
        output = self.attn_drop(output)
        output = self.project(output)
        return output.squeeze()

    def test(self, X):
        output = X.unsqueeze(2)
        output, _ = self.rnn(output)
        output = output.transpose(1, 2)
        output = self.attention(output)
        output = output.transpose(1, 2)
        output = self.project(output)
        output = output.squeeze()
        output = self.activation(output)
        output = torch.argmax(output, dim=1)
        return output
```

4. Result

RFM	Accuracy of fraud/RFM status	Precision of fraud/RFM status	Recall score of fraud/RFM status	F1 score of fraud/RFM status
LSTM	0.878099	0.948529	0.799587	0.867713
CNN	0.872934	0.941320	0.795455	0.862262
逻辑回归模型	0.865702	0.927536	0.793388	0.855233
GaussianNB模型	0.766528	0.916129	0.586776	0.715365
svm模型	0.810950	0.914600	0.685950	0.783943
KNN模型	0.853305	0.880000	0.818181	0.847965
LDA模型	0.844008	0.923664	0.750000	0.827822
随机森林模型	0.887396	0.967581	0.801652	0.876836
Self-attention	0.830578	0.861990	0.787190	0.822894
Lstm_attention	0.462809	0.480686	0.925619	0.632768


RFM
Accuracy of fraud/RFM status
Precision of fraud/RFM status
Recall score of fraud/RFM status
F1 score of fraud/RFM status
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
LSTM
CNN
逻辑…
Gaus…
svm模型
KNN…
LDA…
随机…
Self-…
Lstm_…