Who Should Be Given Incentives? Counterfactual Optimal Treatment Regimes Learning for Recommendation

摘要

有效的个性化激励不仅能提升用户体验，还能增加电子商务平台的收入，从而实现用户与公司之间的双赢局面。传统研究通常通过提升模型来估计用户激励的条件平均处理效应，并在有限预算下通过最大化这些效应的总和进行激励分配。然而，一些用户，无论是否获得激励，依然会进行购买，这些用户被称为“永远购买者”。识别和预测这些“永远购买者”，并减少对他们的激励投放，可以优化激励分配策略。首先，从个体反事实的角度，将用户划分为五个层次，并指出以往提升模型在识别和预测“永远购买者”方面的不足。接着，提出了一种原则性的反事实识别与估计方法，并证明了其无偏性。此外，引入了一种反事实全空间多任务学习方法，以便在有限预算下实现精准的个性化激励策略学习。理论上推导了所学策略的奖励下界，并在三个真实世界的数据集上进行了广泛实验，涵盖了两种常见的激励场景。实验结果表明，所提方法在激励分配上具有显著有效性。

关键词：反事实；最佳处理方案；推荐系统

1 引言

转化反馈强烈反映了用户偏好的信号，并与总商品交易额（GMV）直接相关。为吸引用户兴趣并提升平台收入，许多电子商务公司采用个性化激励措施（如发送优惠券或现金奖励）来提高转化率。这些措施在多种应用场景中得到了广泛应用，例如电子商务交易和音乐网站。针对特定消费者的有效激励机制能够增加用户黏性，实现用户增长，从而使用户和电子商务公司实现双赢。

一般而言，个性化激励政策会利用观察到的用户和物品特征来激励特定的子群体。因此，某些用户可能会接受这些激励，而另一些用户则可能忽略它们。最终，这些激励措施能够促进转化，如图 1中的因果关系图所示。为了有效识别需要奖励的用户，一个重要问题是：“如果向特定用户提供奖励，该用户是否会购买该商品?”然而，在现实场景中，我们无法同时观察到同一用户在有激励和没有激励情况下的转化结果，这就是因果推理中的基本问题。

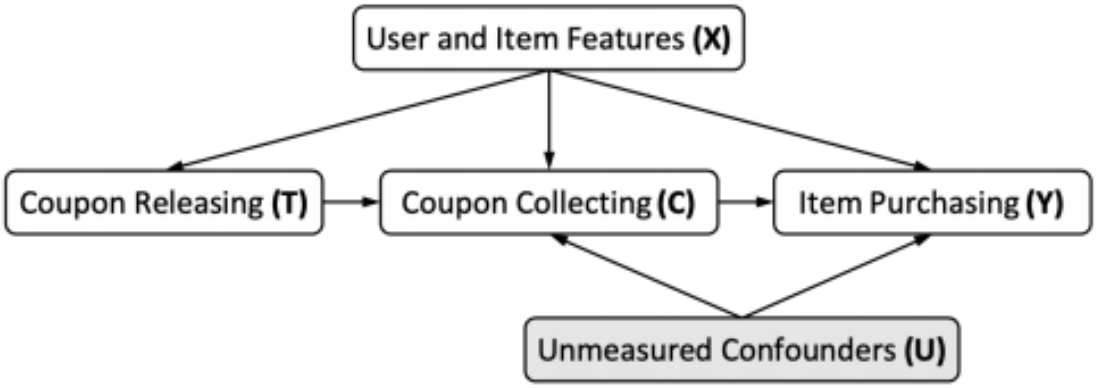

图 1. 电子商务中的因果关系图

为了应对上述问题，最近的研究提出了建模条件平均因果效应（CATEs，亦称为提升建模）的方法 [1]，以识别哪些个体应获得激励。具体而言，如表 1所示，基于 CATE 的方法能够识别“优惠券购买者”，即在接受激励时会积极收集激励并实现转化，而在没有激励时则不会转化的用户。个性化激励算法根据用户和项目的特征，首先估计每个用户作为“优惠券购买者”的概率，然后在有限预算内通过最大化 CATEs 的总和来分配激励。

Strata	Description	C(0)	C(1)	Y(0)	Y(1)	Reward
Y_{0000}	Never Buyer	0	0	0	0	0
Y_{0011}	Never Taker	0	0	1	1	0
Y_{0100}	Coupon Taker	0	1	0	0	0 or -c(x)
Y_{0101}	Coupon Buyer	0	1	0	1	s(x)
Y_{0111}	Always Buyer	0	1	1	1	-c(x)

表 1. 用户-项目对从反事实的角度被划分为五个层次，即 (C(0), C(1), Y(0), Y(1))，分别称为“从不购买者”、“从不领取者”、“优惠券领取者”、“优惠券购买者”和“始终购买者”。

在本文中，如表 2所示，基于 CATE 的方法难以从剩余用户中有效区分“始终购买者”。这些用户无论是否有激励都会进行购买，但在激励存在时，他们会积极收集，从而导致不必要的激励浪费。另一类难以识别的用户是“优惠券领取者”，他们虽然主动接受激励，但却不进行实际转化，这在现金奖励的情况下同样造成资源浪费。与这两类用户不同，“从不购买者”既不主动收集激励，也不进行转化，而“从不领取者”则始终进行转化，但从不主动收集激励。总结而言，“始终购买者”和“优惠券领取者”可能导致激励的浪费，而“从不购买者”和“从不领取者”则不主动收集激励（需要注意的是，这四个子群体对转化的因果效应均为零）。因此，除了通过因果效应激励“优惠券购买者”外，识别并减少对“始终购买者”和“优惠券领取者”的激励，也能有效降低成本并提升平台的收入。

表 2. 对 Naive 方法、条件平均处理效应（CATE）方法和新提出的反事实方法的可识别性进行比较。

Method	Identifiable	Unidentifiable
Naive	N/A	Y_{0000}, Y_{0011}, Y_{0100}, Y_{0101}, Y_{0111}
CATE	Y_{0101}	Y_{0000}, Y_{0011}, Y_{0100}, Y_{0111}
Ours	Y_{0000}, Y_{0011}, Y_{0100}, Y_{0101}, Y_{0111}	N/A

2 相关工作

2.1 提升建模

提升建模旨在估计具有特定特征的个体的条件平均处理效应（CATE），广泛应用于经济学、精准医学、决策制定和广告投放等领域。许多方法已被提出用于估计 CATE，包括结果回归（OR）、逆倾向评分（IPS）以及双重稳健（DR）方法。引入机器学习算法可以进一步提高 CATE 的估计准确性，例如基于树的方法，包括贝叶斯加性回归树（BART）、因果森林（CF），以及基于神经网络的方法，如平衡神经网络（BNN）、反事实回归（CFR）、完美匹配（PM）、X-leraner、DragonNet 和 DESCN 等。然而，这些提升建模方法只能在子群体上估计 CATE，而不能获得个体处理效应。本文旨在扩展上述方法，以识别和估计个体属于每个反事实层次的概率。

2.2 因果推荐

个性化激励（例如，发送优惠券或现金奖励）可以有效提高转化率和总商品交易额（GMV）。以往的数据驱动推荐方法主要依赖关联来预测转化率。然而，这些方法未能有效应对收集数据中的各种偏差和混杂因素，如人气偏差、选择偏差、曝光偏差、从众偏差和位置偏差。为了解决这些问题，已经开发了许多基于因果干预的方法，包括结果回归方法、倾向评分加权方法、双重稳健学习方法和多重稳健学习方法等。引入少量无偏评分还可以进一步提高因果预测的准确性。此外，许多全空间多任务学习方法共享相同的训练空间和推断空间，联合训练模型在经验上也能带来更好的性能，例如全空间多任务模型（ESMM）、多门专家混合模型（MMoE）、Multi-IPW 和 ESCM2。然而，这些方法无法为个体做出反事实预测，而是需要以更细致的方式利用个体的观察结果进行反事实结果预测。尽管相关讨论较少，近期的一些反事实学习研究已致力于生成 Top-N 推荐、缓解点击诱饵问题、估计点击后的转化、消除人气偏差和打破信息茧房等。本文探讨了新颖的个性化激励场景，并扩展了上述因果和多任务学习方法，以进行个体反事实层次预测，从而实现更合理的激励分配。

3 本文方法

3.1 本文方法概述

本文首先使用在因果推理中广泛采用的潜在结果框架对个性化激励情景进行了形式化处理。接着，从反事实的角度将人群划分为五个层次，即基于相同个体的联合潜在结果进行划

分。然后，本文明确指出了以往基于 CATE 的方法的局限性，这些方法只能识别和考虑“优惠券购买者”。随后，本文提出了三种新估计器：反事实结果回归（CF-OR）、反事实逆倾向评分（CF-IPS）和反事实双重稳健估计器（CF-DR），它们能够进一步识别和估算所有五个层次。通过理论分析，本文展示了所提出的 CF-DR 估计器的双稳健性属性，即当填补的结果或学习的倾向性任一准确时，该估计器都是无偏的。

基于提出的反事实识别方法，本文进一步提出了一种名为 CF-MTL 的反事实全空间多任务学习方法，该方法在预算有限的情况下共同训练个体的倾向模型和反事实层次预测模型。与独立训练多个回归模型和倾向模型进行政策学习相比，CF-MTL 能够减轻数据稀疏和偏差放大问题，从而实现更准确的层次预测。本文还理论上推导了所学个性化激励政策的奖励下界。本文在三个真实世界数据集上进行了广泛实验，涵盖了两种常见的激励场景，结果表明，所提出的学习方法能够准确实现个体反事实预测，从而导致显著更具盈利性的激励政策。本文提出的 CF-MTL 模型架构如图 2所示。

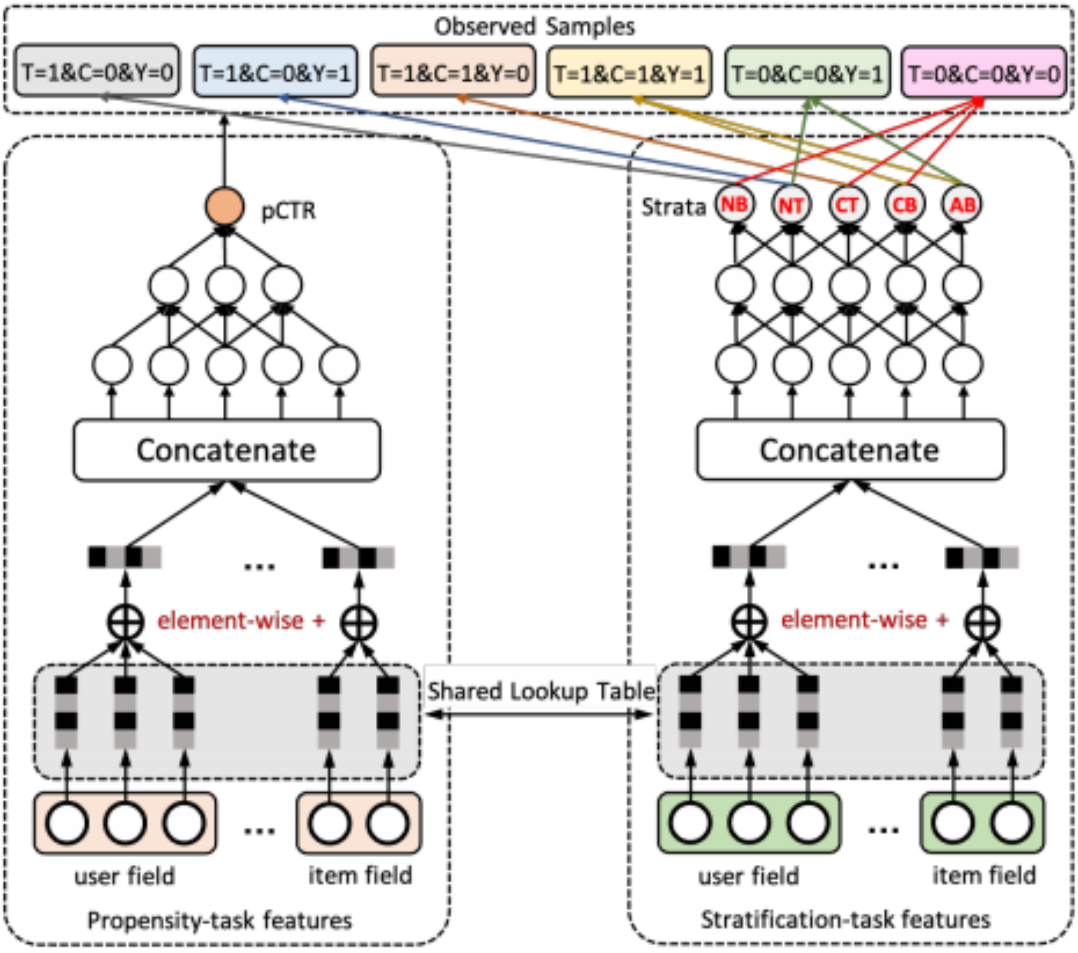

图 2. 提出了反事实全空间多任务学习架构，该架构包含 (i) 倾向模型和 (ii) 个体反事实层预测模型。

3.2 反事实识别方法

从反事实的视角来看，如图 1所示，根据个体的联合潜在结果 $(C(0), C(1), Y(0), Y(1))$ 将所有用户-商品对分为五个层次，并分别命名为“从未购买者”、“从未领取者”、“优惠券领取

者”“优惠券购买者”和“始终购买者”。为了简化，我们将这五组的标签分别表示为 Y_{0000}、Y_{0011}、Y_{0100}、Y_{0101} 和 Y_{1111}。

对于在观测数据中接收到激励 $T=1$ 的单元，从表 1中的 $C(1)$ 列和 $Y(1)$ 列可以发现，只有 Y_{0000} 策略会导致观测结果为 $(T=1,C=0,Y=0)$。类似地，只有 Y_{0011} 策略会导致观测结果为 $(T=1,C=0,Y=1)$，而只有 Y_{0100} 策略会导致观测结果为 $(T=1,C=1,Y=0)$。然而，Y_{0101} 和 Y_{0111} 两种策略都可能导致观测结果为 $(T=1,C=1,Y=1)$。

$$
\begin{aligned}
&P(C=0,Y=0|T=1,X)=P(Y_{0000}|X)\\
&P(C=0,Y=1|T=1,X)=P(Y_{0011}|X)\\
&P(C=1,Y=0|T=1,X)=P(Y_{0100}|X)\\
&P(C=1,Y=1|T=1,X)=P(Y_{0101}|X)+P(Y_{0111}|X)
\end{aligned}
$$

通过类似的论证，对于在观测数据中未以激励 $T=0$ 释放的单元，以下公式成立：

$$
\begin{aligned}
&P(C=1,Y=1|T=0,X)=0\\
&P(C=1,Y=0|T=0,X)=0\\
&P(C=0,Y=1|T=0,X)=P(Y_{0011}|X)+P(Y_{0111}|X)\\
&P(C=0,Y=0|T=0,X)=P(Y_{0000}|X)+P(Y_{0100}|X)+P(Y_{0101}|X)
\end{aligned}
$$

通过关联上诉八个公式，可以识别具有特征 X 的单元属于每个反事实层的概率：

$$
\begin{aligned}
&P(Y_{0000}|X)=P(C=0,Y=0|T=1,X)\\
&P(Y_{0011}|X)=P(C=0,Y=1|T=1,X)\\
&P(Y_{0100}|X)=P(C=1,Y=0|T=1,X)\\
&P(Y_{0101}|X)=P(Y=1|T=1,X)-P(Y=1|T=0,X)\\
&P(Y_{0111}|X)=P(Y=1|T=0,X)-P(C=0,Y=1|T=1,X)
\end{aligned}
$$

3.3 反事实全空间多任务学习方法

3.2 节中，证明了以下公式：

$$
P(C=0,Y=0|T=1,X)=P(Y_{0000}|X)
$$

通过在两边乘以倾向性 $P(T=1|X)$，我们得到：

$$
P(T=1,C=0,Y=0|X)=P(C=0,Y=0|T=1,X)=P(Y_{0000}|X)P(T=1|X)
$$

其中左边是观测 $P(T=1,C=0,Y=0|X)$ 的样本的联合分布，右边包含了反事实层次概率 $P(Y_{0000}|X)$ 和倾向性 $P(T=1|X)$。

设 $f_{0000}(X)$ 为预测 $P(Y_{0000}|X)$ 的模型，$g(x)$ 为预测倾向性 $P(T=1|X)$ 的模型。然后这两个模型可以通过最小化以下损失函数，使用整个空间中的所有用户-项目对来联合训练：

$$
L(f_{0000}(X),g(X),T=1\&C=0\&Y=0)
$$

$L(\cdot,\cdot)$ 是预先指定的损失在所有用户-物品对上的平均值，当样本具有观测值 $T=1, C=0, Y=0$ 时，它等于 1，否则等于 0 ，两个模型通过最小化反事实分层任务损失来联合训练：

$$
\begin{aligned}
&L_s(f_{0000}, f_{0101}, f_{0100}, f_{0101}, f_{0111}; g) \\
&= L(f_{0000}(X)g(X), T=1 \& C=0 \& Y=0) \\
&+ L(f_{0011}(X)g(X), T=1 \& C=0 \& Y=1) \\
&+ L(f_{0100}(X)g(X), T=1 \& C=1 \& Y=0) \\
&+ L(f_{0101}(X)+f_{0111}(X))g(X), T=1 \& C=1 \& Y=1) \\
&+ L((f_{0011}(X)+f_{0111}(X))(1-g(X)), T=0 \& C=0 \& Y=1) \\
&+ L(((f_{0000}(X)+f_{0100}(X)+f_{0101}(X))(1-g(X)), T=0 \& C=0 \& Y=0).
\end{aligned}
$$

此外，为了使倾向性模型 g(X) 准确预测激励传递概率 P(T =1X)，它还通过以下方式进行训练:

$$\mathcal{L}_p(g) = L(g(X), T=1).$$

总的来说，提出的 CF-MTL 方法通过在整个空间内最小化以下损失函数来共同训练 $f(x)$ 和 $g(x)$:

$$L = L_p(g) + \lambda \cdot L_s(f_{000}, f_{001}, f_{0100}, f_{0101}, f_{011}; g)$$

其中，λ 是一个超参数。

给定用户-物品对 (u,i) 属于每个反事实分层的概率，当使用现金奖励作为激励时，奖励应该为 $r_{u,i} = P(Y_{0101}|x_{u,i}) - c(x_{u,i}) \cdot (P(Y_{0101}|x_{u,i}) + P(Y_{0111}|x_{u,i})$；而当使用优惠卷作为激励时，$r_{u,i} = P(Y_{0101}|x_{u,i}) - c(x_{u,i}) \cdot P(Y_{0111}|x_{u,i})$。以优惠券激励为例，最优政策 $\pi^*(x_{u,i}; c)$ 为：

$$\pi^*(\chi_{u,i}; c) = \begin{cases} 1, P(Y_{0101}|x_{ui}) > c(x_{ui}).P(Y_{0111}|x_{ui}) + \gamma(\epsilon; c) \\ d, P(Y_{0101}|x_{ui}) = c(x_{ui}).P(Y_{0111}|x_{ui}) + \gamma(\epsilon; c), \\ 0, P(Y_{0101}|x_{ui}) < c(x_{ui}).P(Y_{0111}|x_{ui}) + \gamma(\epsilon; c) \end{cases}$$

其中 d 是一个介于 0 和 1 之间的值，且 $\gamma(\epsilon; c) >= 0$ 随着预算 ϵ 的增加单调减少。

与基于提升模型的激励放置政策学习相比，反事实策略学习进一步考虑了由” 始终购买者” 引起的额外激励成本 $c(x_{u,i})$。根据预测的概率 $f_{0101}(x_{u,i})$ 和 $f_{0111}(x_{u,i})$，估计的奖励为 $\hat{r_{u,i}} = f_{0101}(x_{u,i}) - c(x_{u,i}) \cdot f_{0111}(x_{u,i})$。然后，通过最大化估计的政策奖励 $\widehat{R}(\pi)$ 并受到预算 ϵ 的限制，来学习反事实个性化激励政策 π^+:

$$\max_{\pi \in \prod} \widehat{R}(\pi) = \frac{1}{|D|} \sum_{(u,i) \epsilon D} \pi(x_{u,i}) \widehat{r}_{u,i}$$

$$s.t. B(\pi) = \frac{1}{|D|} \sum_{(u,i) \epsilon D} \pi(x_{u,i}) \leq \epsilon.$$

4 复现细节

4.1 与已有开源代码对比

源代码已经在 github 上开源，这里以 KuaiRec 数据集上为例，列出修改的代码。

源代码使用一个预训练的 NCF 模型来对原始数据集进行重构，损失函数使用均方误差损失函数，这里修改为二元交叉熵损失函数。

```
class NCF(nn.Module):

    def __init__(self, num_users, num_items, embedding_k=64):
        super(NCF, self).__init__()
        self.num_users = num_users
        self.num_items = num_items
        self.embedding_k = embedding_k
        self.W = torch.nn.Embedding(self.num_users, self.
            embedding_k)
        self.H = torch.nn.Embedding(self.num_items, self.
            embedding_k)
        self.linear_1 = torch.nn.Linear(self.embedding_k*2, 1)
        self.relu = torch.nn.ReLU()
        self.sigmoid = torch.nn.Sigmoid()

        self.xent_func = torch.nn.BCELoss()


    def forward(self, x, is_training=False):
        user_idx = torch.LongTensor(x[:,0]).cuda()
        item_idx = torch.LongTensor(x[:,1]).cuda()
        U_emb = self.W(user_idx)
        V_emb = self.H(item_idx)

        # concat
        z_emb = torch.cat([U_emb, V_emb], axis=1)
        out = self.linear_1(z_emb).squeeze()

        if is_training:
            return self.sigmoid(out), z_emb
        else:
            return self.sigmoid(out)

    def fit(self, x, y, num_epoch=1000, lr=0.05, lamb=0, tol=1e-4,
```

```
        batch_size=128, verbose=0):
33          optimizer = torch.optim.Adam(self.parameters(), lr=lr,
                weight_decay=lamb)
34          last_loss = 1e9
35
36          num_sample = len(x)
37          total_batch = num_sample // batch_size
38
39          early_stop = 0
40          for epoch in range(num_epoch):
41              all_idx = np.arange(num_sample)
42              np.random.shuffle(all_idx)
43              epoch_loss = 0
44              for idx in range(total_batch):
45                  # mini-batch training
46                  selected_idx = all_idx[batch_size*idx:(idx+1)*
                        batch_size]
47                  sub_x = x[selected_idx]
48                  sub_y = torch.Tensor(y[selected_idx]).cuda()
49
50                  optimizer.zero_grad()
51                  pred = self.forward(sub_x, False)
52
53                  xent_loss = self.xent_func(pred, sub_y)
54                  loss = xent_loss
55                  loss.backward()
56                  optimizer.step()
57                  epoch_loss += xent_loss.detach().cpu().numpy()
58
59              relative_loss_div = (last_loss-epoch_loss)/(last_loss+1
                    e-10)
60              if  relative_loss_div < tol:
61                  if early_stop > 5:
62                      print("[NCF] epoch:{}, xent:{}".format(epoch,
                            epoch_loss))
63                      break
64                  early_stop += 1
65
66              last_loss = epoch_loss
67
```

```
            if epoch % 10 == 0 and verbose:
                print("[NCF] epoch:{}, xent:{}".format(epoch,
                    epoch_loss))

            if epoch == num_epoch - 1:
                print("[Warning] Reach preset epochs, it seems does
                    not converge.")
```

对数据集进行重构时，使用 MF 模型生成未观测到的潜在结果，同时在训练本文的反事实全空间多任务学习上，使用 MF 模型预先对用户和物品进行嵌入，经过 one-hot 编码之后映射到低维向量，这里对 MF 的损失函数进行修改，改为二元交叉熵损失函数。

```
class MF(nn.Module):
    def __init__(self, num_users, num_items, embedding_k=4, *args,
        **kwargs):
        super(MF, self).__init__()
        self.num_users = num_users
        self.num_items = num_items
        self.embedding_k = embedding_k
        self.W = torch.nn.Embedding(self.num_users, self.
            embedding_k)
        self.H = torch.nn.Embedding(self.num_items, self.
            embedding_k)

        self.sigmoid = torch.nn.Sigmoid()
        self.xent_func = torch.nn.BCELoss()

    def forward(self, x, is_training=False):
        user_idx = torch.LongTensor(x[:,0]).cuda()
        item_idx = torch.LongTensor(x[:,1]).cuda()
        U_emb = self.W(user_idx)
        V_emb = self.H(item_idx)

        out = self.sigmoid(torch.sum(U_emb.mul(V_emb), 1))

        if is_training:
            return out, U_emb, V_emb
        else:
            return out

    def fit(self, x, y,
        num_epoch=1000, batch_size=128, lr=0.05, lamb=0,
```

```
28          tol=1e-4, verbose=True):
29
30          optimizer = torch.optim.Adam(self.parameters(), lr=lr,
                weight_decay=lamb)
31          last_loss = 1e9
32
33          num_sample = len(x)
34          total_batch = num_sample // batch_size
35
36          early_stop = 0
37          for epoch in range(num_epoch):
38              all_idx = np.arange(num_sample) # 1-6960
39              np.random.shuffle(all_idx)
40              epoch_loss = 0
41
42              for idx in range(total_batch):
43                  # mini-batch training
44                  selected_idx = all_idx[batch_size*idx:(idx+1)*
                        batch_size]
45                  sub_x = x[selected_idx]
46                  sub_y = y[selected_idx]
47                  sub_y = torch.Tensor(sub_y).cuda()
48
49                  pred, u_emb, v_emb = self.forward(sub_x, True)
50
51                  xent_loss = nn.MSELoss()(pred,sub_y)
52
53                  loss = xent_loss
54
55                  optimizer.zero_grad()
56                  loss.backward()
57                  optimizer.step()
58
59                  epoch_loss += xent_loss.detach().cpu().numpy()
60
61              relative_loss_div = (last_loss-epoch_loss)/(last_loss+1
                    e-10)
62              if  relative_loss_div < tol:
63                  if early_stop > 5:
64                      print("[MF] epoch:{}, xent:{}".format(epoch,
```

```
                        epoch_loss))
65                  break
66              early_stop += 1
67
68          last_loss = epoch_loss
69
70          if epoch % 10 == 0 and verbose:
71              print("[MF] epoch:{}, xent:{}".format(epoch,
                    epoch_loss))
72
73          if epoch == num_epoch - 1:
74              print("[MF] Reach preset epochs, it seems does not
                    converge.")
75
76  def predict(self, x):
77      pred, u_emb, v_emb = self.forward(x, True)
78      z_emb = torch.cat([u_emb, v_emb], axis=1)
79      return pred.detach().cpu().numpy(), z_emb.detach().cpu().
            numpy()
```

源代码采用协同训练的方式训练三个子模型 propensity_model、NCF、MLP，损失函数采用 $loss = (alpha1 * L1 + alpha * L2 + beta * L3 + theta * L4 + gamma * L5 + rho * L6) + eta * ctr_loss$，其中 ctr_loss 是给不给予激励 T 预测值和真实值之间的均方误差。这里考虑到总的损失函数 $loss$ 也许会使得 propensity_model 子模型和 NCF 子模型达不到以分开训练时的效果，因此不采用使用同一个损失函数进行协同训练的方式，改为分别使用不同的损失函数进行训练。先对 propensity_model 模型和 NCF 模型进行训练，在将训练之后的模型作为参数传入 MLP 模型中，propensity_model 模型采用均方误差损失函数和 NCF 模型采用二元交叉熵损失函数，MLP 模型损失函数如下面代码所示。

```
1  class MLP(nn.Module):
2      def __init__(self, input_size, propensity_model, NCF_model, *
           args, **kwargs):
3          super().__init__()
4          self.linear_1 = torch.nn.Linear(self.input_size, 5)
5          self.relu = torch.nn.Relu()
6          self.sigmoid = torch.nn.Sigmoid()
7          self.NCF_model = NCF_model
8          self.propensity_model = propensity_model
9
10     def forward(self, x):
11
12         out = nn.Softmax(dim=1)(self.linear_1(x).squeeze())
```

```
13          return out
14
15      def fit(self, x, t, c, y, num_epoch=1000, lamb = 1e-4, tol = 1e
            -4, batch_size=4096, verbose=True, alpha1 = 1, alpha = 1,
            beta = 5, theta = 10, gamma = 5, rho = 5,eta = 1, thr =
            0.05):
16
17          optimizer_MLP = torch.optim.Adam(self.parameters(), lr=lr,
                weight_decay=lamb)
18
19          last_loss = 1e9
20
21          num_sample = len(t)
22          total_batch = num_sample//batch_size
23
24          early_stop = 0
25          for epoch in range(num_epoch):
26              all_idx = np.arange(num_sample)
27              np.random.shuffle(all_idx)
28              epoch_loss = 0
29              for idx in range(total_batch):
30                  selected_idx = all_idx[batch_size*idx:(idx+1)*
                        batch_size]
31                  sub_x = x[selected_idx]
32                  sub_t = t[selected_idx]
33                  sub_c = c[selected_idx]
34                  sub_y = y[selected_idx]
35
36                  pred = self.propensity_model.forward(sub_x, False).
                        squeeze()
37                  pred = torch.clip(pred, thr, 1)
38                  _, pred_class_emb_u, pred_class_emb_v = self.
                        NCF_model.forward(sub_x, True)
39                  pred_class_emb = torch.cat([pred_class_emb_u,
                        pred_class_emb_v], axis = 1)
40                  pred_class = self.forward(pred_class_emb)
41
42                  L1 = F.binary_cross_entropy(pred * pred_class[:, 0]
                         + 1e-6, torch.Tensor(sub_t * (1-sub_c) * (1-
                        sub_y)).cuda(), reduction = 'sum')
```

```
43                  L2 = F.binary_cross_entropy(pred * pred_class[:, 1]
                        + 1e-6, torch.Tensor(sub_t * (1-sub_c) * sub_y)
                        .cuda(), reduction = 'sum')
44                  L3 = F.binary_cross_entropy(pred * pred_class[:, 2]
                        + 1e-6, torch.Tensor(sub_t * sub_c * (1-sub_y))
                        .cuda(), reduction = 'sum')
45                  L4 = F.binary_cross_entropy(pred * (pred_class[:,
                        4] + pred_class[:, 3]) + 1e-6, torch.Tensor(
                        sub_t * sub_c * sub_y).cuda(), reduction = 'sum'
                        )
46                  L5 = F.binary_cross_entropy((1-pred) * (pred_class
                        [:, 4] + pred_class[:, 1]) + 1e-6, torch.Tensor
                        ((1-sub_t) * (1-sub_c) * sub_y).cuda(),
                        reduction = 'sum')
47                  L6 = F.binary_cross_entropy((1-pred) * (pred_class
                        [:, 0] + pred_class[:, 2] + pred_class[:, 3]) +
                        1e-6, torch.Tensor((1-sub_t) * (1-sub_c) * (1-
                        sub_y)).cuda(), reduction = 'sum')
48
49                  loss = (alpha1 * L1 + alpha * L2 + beta * L3 +
                        theta * L4 + gamma * L5 + rho * L6)
50
51                  optimizer_MLP.zero_grad()
52                  loss.backward()
53                  optimizer_MLP.step()
54
55                  epoch_loss += loss.detach().detach().cpu().numpy()
56
57              relative_loss_div = (last_loss-epoch_loss)/(last_loss+1
                    e-10)
58              if  relative_loss_div < tol:
59                  if early_stop > 5:
60                      print("[MLP] epoch:{}, xent:{}".format(epoch,
                            epoch_loss))
61                      break
62                  early_stop += 1
63
64              last_loss = epoch_loss
65
66              if epoch % 10 == 0 and verbose:
```

```
67                 print("[MLP] epoch:{}, xent:{}".format(epoch,
                       epoch_loss))
68
69             if epoch == num_epoch - 1:
70                 print("[Warning] Reach preset epochs, it seems does
                       not converge.")
71
72     def predict(self, x):
73         _, pred_class_emb_u, pred_class_emb_v = self.NCF_model.
               forward(x, True)
74         pred_class_emb = torch.cat([pred_class_emb_u,
               pred_class_emb_v], axis=1)
75         pred_class =  nn.Softmax(dim=1)(self.linear_1(
               pred_class_emb).squeeze())
76         return pred_class.detach().cpu().numpy().flatten()
77
78 thr = 0.1
79 propensity_model = NCF(num_user, num_item, embedding_k =
       embedding_k)
80 propensity_model.cuda()
81 propensity_model.fit(x_tr,t_tr,lamb = 1e-5,lr= 0.01,batch_size =
       8192, verbose=True)
82 pred = propensity_model.forward(x_tr, False).squeeze()
83 pred = torch.clip(pred, thr, 1)
84 NCF_model = MF(num_user, num_item, embedding_k = embedding_k)
85 NCF_model.cuda()
86 NCF_model.fit(x_tr,y_tr,lamb = 1e-5,lr = 0.01,batch_size = 8192,
       verbose=True)
87 _, pred_class_emb_u, pred_class_emb_v = NCF_model.forward(x_tr,
       True)
88 pred_class_emb = torch.cat([pred_class_emb_u, pred_class_emb_v],
       axis=1)
89 MLP_model = MLP(input_size = embedding_k * 2, propensity_model =
       propensity_model, NCF_model = NCF_model)
90 MLP_model.cuda()
91 MLP_model.fit(x_tr, t_tr, c_tr, y_tr, eta = 0.5, alpha1 = 1, alpha
       = 1, beta = 1, theta = 5, gamma = 10, rho = 10, thr = 0.1, lr
       =0.01, batch_size=8192, lamb=1e-5, tol=1e-5, verbose=True)
92
93
```

```
if scenario == 'cash':
cost = np.array([0, 0, -C, 1, -C])

if scenario == 'coupon':
cost = np.array([0, 0, 0, 1, -C])

pred_class = MLP_model.predict(constructed_data_test[:, :2]).
    reshape([-1,5])
reward_ours = np.sum(pred_class * cost, axis=1)

temp_ours = np.c_[constructed_data_test, reward_ours]
temp_ours = temp_ours[np.argsort(-temp_ours[:, -1])]

temp_5 = temp_ours[:,min(budget, sum(temp_ours[:, -1] >  )/
    temp_ours.shape[ ])* temp_ours.shape[ ]]
a = sum(temp_5[:, 6])== 1)
b = sum(temp_5[:, 6])== 2)
c = sum(temp_5[:, 6])== 3)
d = sum(temp_5[:, 6])== 4)
e = sum(temp_5[:, 6])== 5)
Real_reward = sum(temp_5[:, -2])
print(a, b, c, d, e, Real_reward)
```

4.2 实验环境搭建

为了运行以上修改之后的代码，需要预安装几个 python 库 (numpy、pandas、pytorch、sklearn),实验使用的版本为:python(3.10.15)、pandas(2.2.3)、pytorch(2.5.1+cu118)、sklearn(1.5.2)、numpy(1.26.3)。

4.3 数据集与预处理

为了评估所提出的反事实估计和政策学习方法，本文在三个真实世界数据集上进行了广泛的实验:Yep、ML-1M 和 KuaiRec。这些数据集均为公开可用，具有不同的领域、规模和稀疏性，统计信息汇总在表 3中。根据以往研究的做法，对于 Ye0 和 ML-1M 数据集，我们将观察到的评分进行二值化处理，评分大于 3 的标记为 1, 其余标记为 0，作为结果变量 Y，而评分观察指标则作为结果变量 C。KuaiRec 是一个来自短视频分享平台的完全曝光数据集，我们随机选择 20% 的样本作为观察值 C =1，并将观看比例超过 0.6 的视频标记为 1，否则标记为 0。

接下来，我们为每个单位执行反事实层次标记。需要注意的是，对于所有数据集，观察到的单位 (C =1,Y = 0) 必须被标记为” 优惠券使用者”。为了标记其余单位，我们预训练了一个神经协同过滤 (NCF) 模型来生成预测评分。具体而言，对于观察到的单位 (C = 1,Y = 1),

数据集	#Users	#Items
YELP	25,677	25,815
ML-1M	6,040	3,952
KUAIREC	1,411	3,327

表 3. 数据集总结

我们将预测评分最高的单位的一半标记为 “常购者”，其余一半标记为 “优惠券购者”。前者的结果总是 Y(0)= Y(1)= 1，且其观察评分平均值相对较高，类似地，对于缺失评分的单位 (即 C = 0)，我们将预测评分最高的一半标记为 “绝不使用者”，其余一半标记为 “绝不购者”。

4.4 创新点

- 本文从个性化反事实的角度重新定义了个性化激励政策学习问题，揭示了以往提升建模方法的局限性，并提出了原则性的反事实估计器，用于识别和估计个体属于特定反事实层次的概率。
- 本文基于提出的反事实识别方法，进一步提出了一种反事实全空间多任务学习方法，，该方法在预算有限的情况下共同训练个体的倾向模型和反事实层次预测模型，以准确执行个性化激励政策学习。此外，还理论推导了学习政策奖励的下界。

5 实验结果分析

Coupon	YELP				ML-1M				KUAIREC			
Methods	Positive	Negative	Neutral	Reward	Positive	Negative	Neutral	Reward	Positive	Negative	Neutral	Reward
Naive	35,527	31,781	90,656	22,814	51,153	34,315	130,577	37,426	3,358	57,670	141,000	-19,710
OR	57,911	27,765	72,297	46,804	77,018	45,017	94,010	59,011	38,172	**7,942**	47,077	34,995
CF-OR	**58,155**	**22,963**	76,855	**48,969**	**78,878**	**40,300**	96,876	**62,757**	**70,671**	22,419	108,938	**61,703**
IPS	**56,177**	25,833	75,963	45,843	80,574	42,708	92,763	63,490	77,585	19,111	105,332	69,940
CF-IPS	55,838	**22,615**	79,520	**46,791**	**81,371**	**35,986**	98,688	**66,976**	**79,816**	**17,761**	104,451	**72,711**
DR	57,872	27,201	72,900	46,991	78,547	44,418	93,080	60,779	71,578	**10,097**	122,032	67,539
CF-DR	**57,971**	**22,050**	77,952	**49,150**	**80,103**	**38,759**	97,183	**64,599**	**81,104**	15,867	105,057	**74,757**
CF-MTL	**66,963**	13,630	77,560	61,510	85,850	34,078	96,117	72,218	90,387	**12,184**	99,457	85,513
CF-MTL-v1	66,931	**11,932**	79,099	**62,158**	**86,094**	**21,133**	108,818	**77,640**	**90,668**	12,400	98,960	**85,708**
Cash	YELP				ML-1M				KUAIREC			
Methods	Positive	Negative	Neutral	Reward	Positive	Negative	Neutral	Reward	Positive	Negative	Neutral	Reward
Naive	35,527	67,569	54,877	8,499	51,153	89,765	75,127	15,246	3,358	108,732	89,938	-40,134
OR	**57,911**	51,953	48,109	37,129	**77,018**	106,710	32,317	34,333	38,172	**13,782**	41,237	32,659
CF-OR	56,287	**41,945**	59,741	**39,508**	76,748	**90,987**	48,310	**40,353**	**67,652**	44,659	89,717	**49,788**
IPS	**56,177**	49,435	52,361	36,402	**80,574**	92,415	43,056	43,607	77,585	34,430	90,013	63,812
CF-IPS	54,228	**42,395**	61,350	**37,269**	79,051	**69,800**	67,194	**51,130**	**77,665**	**34,380**	89,983	**63,912**
DR	**57,872**	50,951	49,150	37,491	**78,547**	101,051	36,447	38,126	71,578	**20,755**	73,974	63,275
CF-DR	55,869	**40,194**	61,910	**39,791**	77,646	**80,770**	57,629	**45,337**	**80,044**	32,160	89,824	**67,180**
CF-MTL	67,025	25,014	65,934	57,019	**85,866**	58,477	71,702	62,475	90,514	21,618	89,896	81,866
CF-MTL-v1	**67,032**	**23,170**	67,760	**57,763**	**85,649**	**37,754**	92,642	**70,547**	**90,686**	**21,549**	89,793	**82,066**

表 4. 源代码运行结果

通过分析上面的表格，可以发现，修改之后的代码更能够识别个体属于 negative 的样本，从而在有限预算下减少花费。在有限的预算下，减少的花费又会进一步用于识别个体属于 positive 的样本，从而提高获得的 reward。

同时，利用消融实验验证所提出的 CF-MTL 方法的有效性，通过改变训练损失显示了两种激励场景下的 YELP 和 ML-1M 的奖励。表 5 和表 6 显示了两种激励场景下的 YELp 和 ML-1M 的奖励。从理论上讲，当 L_{001} 和 L_{000} 都被移除时，CF-MTL 无法识别 “优惠券购买者” 和 “始终购买者”。因此，这种方法在所有场景和数据集上的性能显著变差。

Methods	L_{001}	L_{000}	$\mathbf{Reward_{Coupon}}$	$\mathbf{Reward_{Cash}}$
CF-MTL w/o L_{001} L_{000}	×	×	15,913	8,445
CF-MTL w/o L_{000}	✓	×	60,186	55,417
CF-MTL w/o L_{001}	×	✓	56,965	52,373
CF-MTL	✓	✓	**62,158**	**57,763**

表 5. YELP 数据集训练损失消融实验。

Methods	L_{001}	L_{000}	$\mathbf{Reward_{Coupon}}$	$\mathbf{Reward_{Cash}}$
CF-MTL w/o L_{001} L_{000}	×	×	44,163	35,232
CF-MTL w/o L_{000}	✓	×	75,273	67,789
CF-MTL w/o L_{001}	×	✓	73,363	65,107
CF-MTL	✓	✓	**77,640**	**70,547**

表 6. ML-1M 数据集训练损失消融实验。

6 总结与展望

本文从个性化反事实的角度重新定义了个性化激励政策学习问题，并提出了原则性的反事实估计器，用于识别和估计个体属于特定反事实层次的概率，并对全空间多任务学习方法进行改进，通过识别优惠券与购买者之间的因果关系，提高了获得的 reward。

在实现过程中，在重构数据集的过程中，发现利用神经协同过滤网络预训练的个体反事实预测器，预测个体反事实与真实情况准确率仅为 80%，有望在未来提出更准确的预测器，提高预测准确率，进一步减少花费，提高获得得 reward。

参考文献

[1] Masahiro Sato, Janmajay Singh, Sho Takemori, Takashi Sonoda, Qian Zhang, and Tomoko Ohkuma. Uplift-based evaluation and optimization of recommenders. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 296–304, 2019.