第5章 基于深度学习的推荐

5.1 基于行为的协同过滤

协同过滤是一种在推荐系统中广泛采用的推荐方法。这种算法基于一个“物以类聚，人以群分”的假设，喜欢相同物品的用户更有可能具有相同的兴趣。基于协同过滤的推荐系统一般应用于有用户评分的系统之中，通过分数去刻画用户对于物品的喜好。协同过滤被视为利用集体智慧的典范，不需要对项目进行特殊处理，而是通过用户建立物品与物品之间的联系。 目前，协同过滤推荐系统被分化为两种类型：基于用户（user-based）的推荐和基于物品（Item-based）的推荐。

1. 基于用户的协同过滤

将目标用户对项目的历史评价与其他用户匹配，找到相似用户，再将相似用户感兴趣的项目推荐给目标用户。基于用户的协同过滤推荐的基本原理：根据所有用户对物品或者信息偏好（评分），发现与当前用户口味和偏好相似的“邻居”用户群，在一般应用中是采用计算 K 近邻的算法；基于这 K 个邻居的历史偏好信息，为当前用户进行推荐。这种推荐系统的优点在于推荐物品之间在内容上可能完全不相关，因此可以发现用户的潜在兴趣，并且针对每个用户生成其个性化的推荐结果。其缺点是，一般的 Web 系统中，用户的增长速度都远远大于物品的增长速度，因此其计算量的增长巨大，系统性能容易成为瓶颈。因此，在业界中单纯的使用基于用户的协同过滤系统较少。

2. 基于项目（物品）的协同过滤

基于项目（物品）的协同过滤是指利用项目间的相似性，而非用户间的相似性来计算预测值，从而实施推荐。基于物品的协同过滤和基于用户的协同过滤相似，它使用所有用户对物品或者信息的偏好（评分），发现物品和物品之间的相似度，然后根据用户的历史偏好信息，将类似的物品推荐给用户。基于物品的协同过滤可以看作关联规则推荐的一种退化，但由于协同过滤更多考虑了用户的实际评分，并且只是计算相似度而不是寻找频繁集，因此可以认为基于物品的协同过滤准确率较高并且覆盖率更高。同基于用户的推荐相比，基于物品的推荐应用更为广泛，扩展性和算法性能更好。由于项目的增长速度一般较为平缓，因此性能变化不大。其缺点就是无法提供个性化的推荐结果。

3. 协同过滤流程

（1）依据行为记录挖掘用户偏好特征，构建用户画像。

（2）根据评分数据集，进行相似度计算，为用户或项目寻找最近邻集合。

（3）根据最近邻集合，预测用户对项目的评分，设置一个阈值或是直接取前几项，构建候选推荐集。

5.2 基于深度学习的推荐

基于深度学习的推荐是将深度学习技术融合在传统的推荐算法（如基于内容的推荐、协同过滤推荐）之中，或使用无监督学习方法对项目进行聚类，或使用监督学习方法对项目进行分类，以及使用多层感知器、卷积神经网络、循环神经网络、递归神经网络等对数据加工处理提取特征。基于深度学习主要是体现在使用机器学习的数据处理技术，通过组合低层特征形成更加稠密的高层语义抽象，从而自动发现数据的分布式特征表示，解决了传统机器学习中需要人工设计特征的问题。深度学习技术是要依托于传统推荐技术的，可以说是对传统推荐技术的增强。该类型推荐多用于处理图像、文本、音频等数据。如电子商务平台、电影售票系统等，主营项目都会附带明显的图片介绍，可以根据用户当前浏览或是历史购买记录来获取图片信息，深度学习提取出图像的特征表示，再以此从项目数据

库中比对类似特征的图像，从而进行推荐。像亚马逊这样的网上书店或小说平台，主营项目以文本信息为主。经过深度学习，也可以提取出文本的风格、类型、特色等特征，从而进行匹配推荐。对于音乐播放器这类的以音频为主的系统，先将音频数据变为数字信号，再进行深度学习，用数字信息抽象表示音频特征（舒缓、嘻哈、古典等），从而可训练出用户的听曲风格。基于深度学习推荐的最大优势就是针对多种类型的输入数据，都可以提取特征，并训练模型，可以实现多元化的推荐，但是要想得到更好的推荐效果，就需要更长的时间来训练模型。

基于深度学习的推荐系统中的常用神经网络如下。

卷积神经网络包括输入层、卷积层、池化层、全连接层和输出层，其中卷积层和池化层组合形成了特征提取器。在卷积层中，上一个神经元不再与全部的下一层神经元全连接，只是部分连接，并且在 CNN 中采用了权值共享，即卷积核，这不仅有效减少了神经网络中的参数个数，还降低了过拟合的概率。卷积神经网络多用于处理图像数据，所以经常是通过处理分析用户项目的历史图片信息来推荐类似风格和颜色布局图片的其他项目。

循环神经网络相较于普通神经网络，其特殊之处在于它各个隐藏层之间是具有连接的，体现在功能上就是能够记忆之前的信息，即在当前隐藏层的输入中不仅包括输入层的输出信息，还包括隐藏层上一个状态（或上一个时刻）的输出。这种神经网络多用于处理序列数据，如语音识别，要想语义翻译准确，就要根据上文环境进行判断，所以循环神经网络在处理这类问题时就具有一定的优势。

1. DeepFM 的背景

DeepFM 是在 FM（factorization machines，因子分解机）算法的基础上衍生出来的算法其模型结构，如图 5-1 所示。

DeepFM 将 FM 与 DNN 相结合，联合训练 FM 模型和 DNN 模型，用 FM 做特征间的低阶组合，用 DNN 做特征间的高阶组合。相比于谷歌最新发布的 Wide&Deep 模型，DeepFM 模型的 Deep component 和 FM component 从 Embedding 层共享数据输入，同时不需要专门的特征工程。

DeepFM 广泛应用于 CTR（click through rate，点击通过率）预估领域，通过用户的点击行为来学习潜在的特征交互在 CTR 中至关重要。隐藏在用户点击行为背后的特征交互，无论是低阶交互还是高阶交互都可能对最终的 CTR 产生影响。FM 算法可以对特征间成对

的特征交互以潜在向量内积的方式进行建模，并表现出不错的效果。然而，FM 由于高复杂性不能进行高阶特征交互，常用的 FM 特征交互通常局限于二阶。其他的基于神经网络的特征交互的方法要么侧重于低阶或者高阶的特征交互，要么依赖于特征工程，因此，DeepFM 出现了。DeepFM 表明，通过一个端到端的方式学习所有阶特征之间的交互并且不严格依赖特征工程也是可行的。

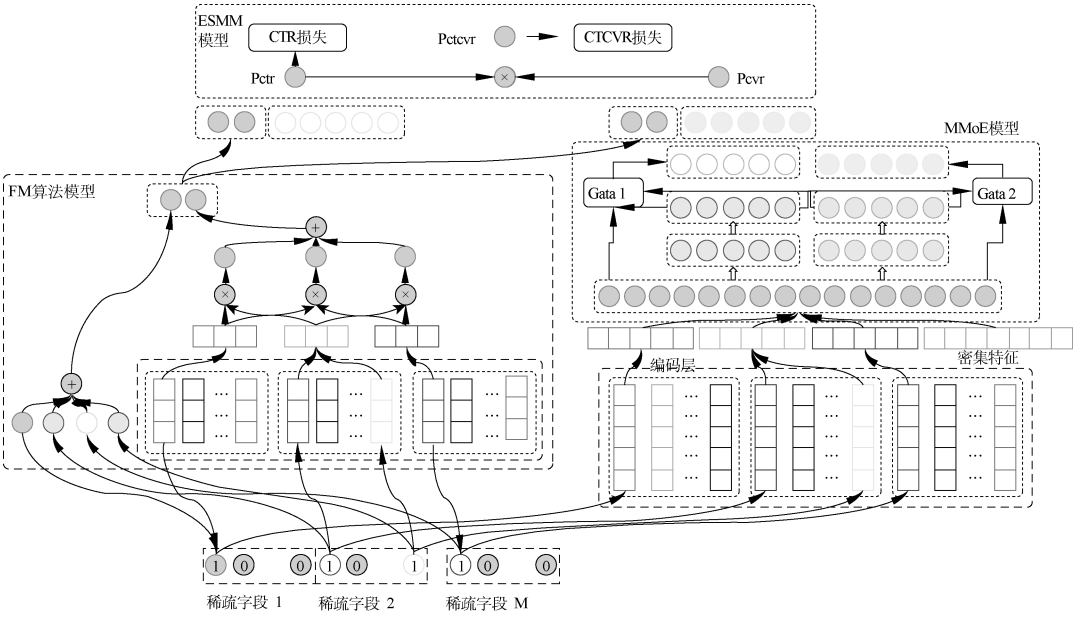

图 5-1　DeepFM 模型

2. DeepFM 的特点

DeepFM 是一个结合了 FM 结构和 DNN 结构的新的神经网络模型，并且 DeepFM 能够像 FM 那样进行低阶特征间的交互，也能够像 DNN 那样进行高阶特征间的交互。同时，DeepFM 多层网络模型能够进行端到端的训练且不依赖于特征工程，如图 5-2 所示。其主要特点分析如下。

☑ 共享输入：DeepFM 的 FM component 和 Deep component 共享相同的输入，因此能够完成高效训练。

☑ 输入层（sparse features）：输入数据包括类别特征和连续特征。

☑ Embedding 层（dense embeddings）：该层的作用是对类别特征进行 Embedding 向量

化，将离散特征映射为稠密特征。该层的结果同时被提供给 FM Layer 和 Hidden Layer，即 FM Layer 和 Hidden Layer 共享相同的 Embedding 层。

☑ FM 层（FM layer)：该模型主要提取一阶特征和两两交叉特征。

☑ 隐藏层（hidden layer)：该模块主要是应用 DNN 模型结构提取深层次的特征信息。

☑ 输出单元（output units)：对 FM Layer 和 Hidden Layer 的结果执行 Sigmoid 函数，得出最终的结果。

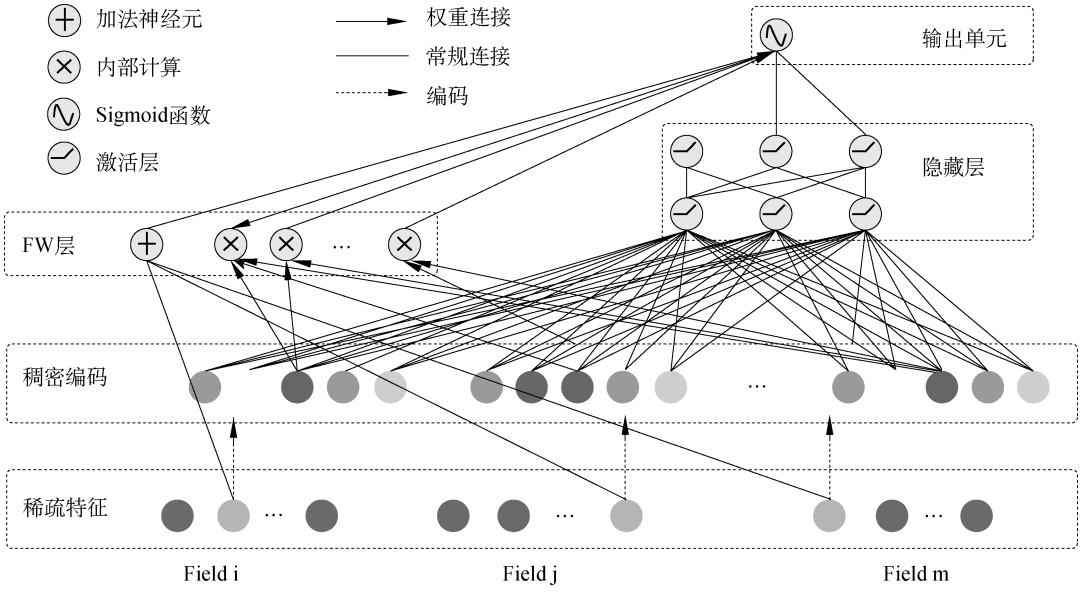

图 5-2　DeepFM 多层网络模型

3. 输入层

DeepFM 的输入可由连续型变量和类别型变量共同组成，且类别型变量需要进行 One-Hot 编码。正是由于 One-Hot 编码导致了输入特征变得高维且稀疏。针对高维稀疏的输入特征，DeepFM 采用了 word2vec 的词嵌入（wordembedding）思想，把高维稀疏的向量映射到相对低维且向量元素都不为零的空间向量中，不同的是 DeepFM 根据特征类型进行了 field 区分，即将特征分为不同的 field。

在处理特征时，我们需要对离散型数据进行 one-hot 转化，经过 one-hot 之后，一列会变成多列，这样会导致特征矩阵变得非常稀疏。

4. embedding 层

embedding 层对类别特征进行 embedding 向量化，将离散特征映射为稠密特征。embedding 层的输入就是分 field 的特征，也就是说 embedding 层完成了对不同特征按 field 进行向量化。FM 层和 hidden 层共享的就是 embedding 层的输出结果。

5. FM 层

FM 层的输入是 embedding 层的输出，FM 层主要是提取一阶特征和两两交叉的二阶特征。如图 5-2 所示，Field_i、Field_j、Field_m 中的黄色圆点指向 Addition 节点的黑线表示的是 FM 直接对原始特征做的一阶计算。而 embedding 层每个 field 对应的 embedding 会有两条红线连接到 Inner Product 节点，表示的是 FM 对特征进行的二阶交叉计算。

6. 隐藏层

hidden layer 主要是应用 DNN 的模型结构提取深层次的特征信息。hidden layer 的输入也是 embedding 层的输出（与 FM layer 共享输入）。从 embedding 层输出到 hidden layer 是一种全连接计算。

7. 输出层

输出层主要对 FM layer 和 hidden layer 的结果进行 Sigmoid 操作，得出最终的结果。

8. FM Component

DeepFM 中的 FM 部分是一个因子分解机，除了所有特征间的一个线性组合（一阶），FM 模型也支持以独立特征向量内积形式的成对特征组合（二阶）。相比于先前的方法，FM 在处理二阶特征组合的时更有效，尤其在训练数据集是稀疏的场景时。在先前的方法中，特征 i 和特征 j 的组合参数只有在特征 i 和特征 j 同时出现在相同的数据记录中时才能得到训练。然而在 FM 中，这个参数可以通过向量 Vi 和向量 Vj 的内积的形式完成更新。这样，FM 能够训练 Vi（Vj）无论 i（j）是否出现在数据记录中。这样很少出现在训练集中的特征组合也能够被 FM 很好地学习出来。

9. Deep Component

DeepFM 中的 deep 部分是一个前馈神经网络，用来学习高阶特征组合。DeepFM 中的 deep 部分将一条向量输入到神经网络。通常，点击预估任务的神经网络输入要求网络结

构的设计。点击预估的原始特征输入向量通常是高度稀疏的、超高维、连续值与绝对值混合、按 fields 分组的形式，这就需要网络中有一个嵌入层（embedding layer）在将向量输入第一个 hidden 层之前将输入向量压缩成一个低维、实值稠密的向量，否则网络将难于训练。

10. DNN 部分的网络结构

☑ 输入层：输入数据包括类别特征和连续特征。

☑ Embedding 层：该层的作用是对类别特征进行 embedding 向量化，将离散特征映射为稠密特征。

☑ 隐藏层：该模块主要是应用 DNN 模型结构，提取深层次的特征信息。

☑ 输出层：对 FM layer 的结果进行 Sigmoid 操作，得出最终的结果。

如图 5-3 所示，DeepFM 将 FM 模型和 DNN 模型都当作全面学习的网络结构，这一点跟一些其他方法中通过预训练 FM 的隐藏向量进而对网络进行初始化的方式有些不同。这样的方法可以消除对 FM 的预训练，并且可以通过端到端的方式完成对整个网络的联合训练。

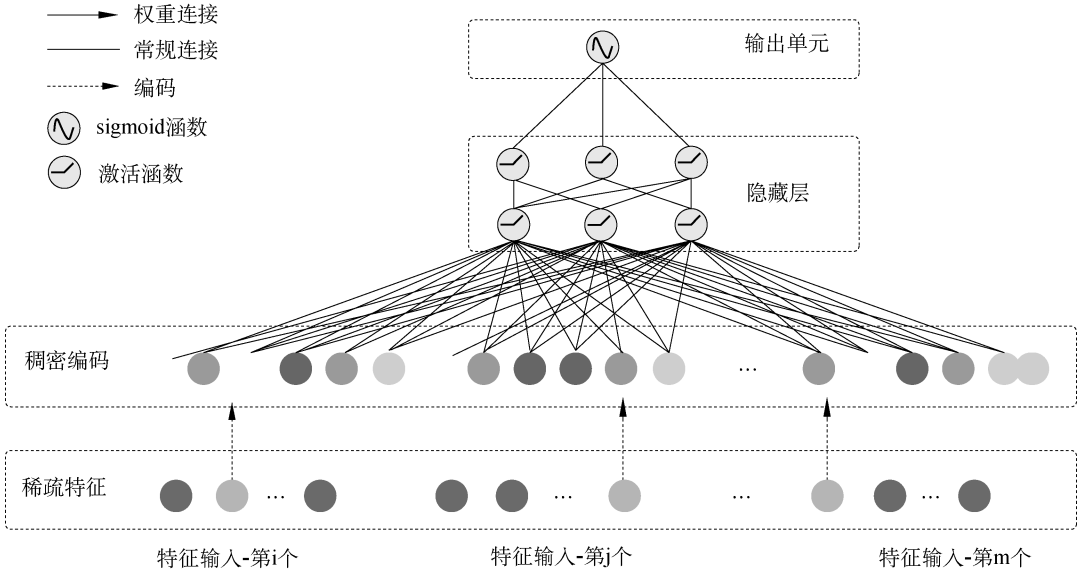

图 5-3　DeepFM 部分网络模型

11. DeepFM 的一些网络参数设置

Activation Function：相比于 sigmoid，relu、tanh 更适合 deep 模型。

Dropout：Dropout 影响一个神经元被保留在网络中的概率，Dropout 是折中精度和网络复杂度的一种正则化技术。

Number of Neurons per Layer：增加每层神经元的个数可能造成网络更复杂，复杂的模型容易过拟合。

Number of Hidden Layers：增加隐藏层的数量在模型开始训练的时候会提升训练效果，但是如果隐藏层的数量一直增加可能会造成训练效果下降，这也是一种过拟合现象。

5.3 基于 Pytorch 的 DeepFM 的完整实战代码

1. 导入必要的包

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.utils.data as Data

import time, json, datetime
from tqdm import tqdm

import numpy as np
import pandas as pd
from sklearn.metrics import log_loss, roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
```

2. 定义 DeepFM 模型

```
class DeepFM(nn.Module):
```

```
    def __init__(self, cate_fea_nuniqs, nume_fea_size=0, emb_size=8,
                 hid_dims=[256, 128], num_classes=1, dropout=[0.2, 0.2]):
        """
        cate_fea_nuniqs: 类别特征的唯一值个数列表，也就是每个类别特征的 vocab_size
所组成的列表
        nume_fea_size: 数值特征的个数，该模型会考虑到输入全为类别型，即没有数值特征
的情况
        """
        super().__init__()
        self.cate_fea_size = len(cate_fea_nuniqs)
        self.nume_fea_size = nume_fea_size
        """FM 部分"""
        # 一阶
        if self.nume_fea_size != 0:
            self.fm_1st_order_dense = nn.Linear(self.nume_fea_size, 1)  #
数值特征的一阶表示
        self.fm_1st_order_sparse_emb = nn.ModuleList([
            nn.Embedding(voc_size, 1) for voc_size in cate_fea_nuniqs])  #
类别特征的一阶表示
        self.fm_2nd_order_sparse_emb = nn.ModuleList([
            nn.Embedding(voc_size, emb_size) for voc_size in cate_fea_
nuniqs])  # 类别特征的二阶表示

        """DNN 部分"""
        self.all_dims = [self.cate_fea_size * emb_size] + hid_dims
        self.dense_linear = nn.Linear(self.nume_fea_size, self.cate_fea_
size * emb_size)  # 数值特征的维度变换到与 FM 输出维度一致
        self.relu = nn.ReLU()
        # for DNN
        for i in range(1, len(self.all_dims)):
            setattr(self, 'linear_'+str(i), nn.Linear(self.all_dims[i-1],
self.all_dims[i]))
            setattr(self, 'batchNorm_' + str(i), nn.BatchNorm1d(self.all_
dims[i]))
            setattr(self, 'activation_' + str(i), nn.ReLU())
            setattr(self, 'dropout_'+str(i), nn.Dropout(dropout[i-1]))
        # for output
```

```
        self.dnn_linear = nn.Linear(hid_dims[-1], num_classes)
        self.sigmoid = nn.Sigmoid()

    def forward(self, X_sparse, X_dense=None):
        """
        X_sparse: 类别型特征输入  [bs, cate_fea_size]
        X_dense: 数值型特征输入（可能没有）  [bs, dense_fea_size]
        """
        """FM一阶部分"""
        fm_1st_sparse_res = [emb(X_sparse[:, i].unsqueeze(1)).view(-1, 1)
                             for i, emb in enumerate(self.fm_1st_order_sparse_
emb)]
        fm_1st_sparse_res = torch.cat(fm_1st_sparse_res, dim=1)   # [bs,
cate_fea_size]
        fm_1st_sparse_res = torch.sum(fm_1st_sparse_res, 1, keepdim=True)
# [bs, 1]

        if X_dense is not None:
            fm_1st_dense_res = self.fm_1st_order_dense(X_dense)
            fm_1st_part = fm_1st_sparse_res + fm_1st_dense_res
        else:
            fm_1st_part = fm_1st_sparse_res   # [bs, 1]

        """FM二阶部分"""
        fm_2nd_order_res = [emb(X_sparse[:, i].unsqueeze(1)) for i, emb in
enumerate(self.fm_2nd_order_sparse_emb)]
        fm_2nd_concat_1d = torch.cat(fm_2nd_order_res, dim=1)   # [bs, n,
emb_size]  n为类别型特征个数(cate_fea_size)

        # 先求和再平方
        sum_embed = torch.sum(fm_2nd_concat_1d, 1)  # [bs, emb_size]
        square_sum_embed = sum_embed * sum_embed    # [bs, emb_size]
        # 先平方再求和
        square_embed = fm_2nd_concat_1d * fm_2nd_concat_1d   # [bs, n,
emb_size]
        sum_square_embed = torch.sum(square_embed, 1)  # [bs, emb_size]
        # 相减除以2
```

```
        sub = square_sum_embed - sum_square_embed
        sub = sub * 0.5   # [bs, emb_size]

        fm_2nd_part = torch.sum(sub, 1, keepdim=True)   # [bs, 1]

        """DNN 部分"""
        dnn_out = torch.flatten(fm_2nd_concat_1d, 1)   # [bs, n * emb_size]

        if X_dense is not None:
            dense_out = self.relu(self.dense_linear(X_dense))   # [bs, n *
emb_size]
            dnn_out = dnn_out + dense_out   # [bs, n * emb_size]

        for i in range(1, len(self.all_dims)):
            dnn_out = getattr(self, 'linear_' + str(i))(dnn_out)
            dnn_out = getattr(self, 'batchNorm_' + str(i))(dnn_out)
            dnn_out = getattr(self, 'activation_' + str(i))(dnn_out)
            dnn_out = getattr(self, 'dropout_' + str(i))(dnn_out)

        dnn_out = self.dnn_linear(dnn_out)   # [bs, 1]
        out = fm_1st_part + fm_2nd_part + dnn_out   # [bs, 1]
        out = self.sigmoid(out)
        return out
```

3. 读入数据并进行预处理

数据是 criteo 数据集（比较经典的点击率预估数据集）的随机 50w 样本。

```
data = pd.read_csv("criteo_sample_50w.csv")

dense_features = [f for f in data.columns.tolist() if f[0] == "I"]
sparse_features = [f for f in data.columns.tolist() if f[0] == "C"]

data[sparse_features] = data[sparse_features].fillna('-10086', )
data[dense_features] = data[dense_features].fillna(0, )
target = ['label']

## 类别特征 labelencoder
```

```
for feat in tqdm(sparse_features):
    lbe = LabelEncoder()
    data[feat] = lbe.fit_transform(data[feat])

## 数值特征标准化
for feat in tqdm(dense_features):
    mean = data[feat].mean()
    std = data[feat].std()
    data[feat] = (data[feat] - mean) / (std + 1e-12)   # 防止除零

print(data.shape)
data.head()
```

4. 定义 dataloader、模型和优化器等

```
train, valid = train_test_split(data, test_size=0.2, random_state=2020)
print(train.shape, valid.shape)

train_dataset = Data.TensorDataset(torch.LongTensor(train[sparse_features].
values),

torch.FloatTensor(train[dense_features].values),

torch.FloatTensor(train['label'].values),)

train_loader  =  Data.DataLoader(dataset=train_dataset,  batch_size=2048,
shuffle=True)

valid_dataset  =  Data.TensorDataset(torch.LongTensor(valid[sparse_features].
values),

torch.FloatTensor(valid[dense_features].values),
                              torch.FloatTensor(valid['label'].values),)
valid_loader  =  Data.DataLoader(dataset=valid_dataset,  batch_size=4096,
shuffle=False)
device   =   torch.device('cuda')   if   torch.cuda.is_available()   else
torch.device('cpu')
print(device)
cate_fea_nuniqs = [data[f].nunique() for f in sparse_features]
```

```
model = DeepFM(cate_fea_nuniqs, nume_fea_size=len(dense_features))
model.to(device)
loss_fcn = nn.BCELoss()  # Loss 函数
loss_fcn = loss_fcn.to(device)
optimizer = optim.Adam(model.parameters(), lr=0.005, weight_decay=0.001)
scheduler   =   torch.optim.lr_scheduler.StepLR(optimizer,   step_size=1,
gamma=0.8)

# 打印模型参数
def get_parameter_number(model):
    total_num = sum(p.numel() for p in model.parameters())
    trainable_num   =   sum(p.numel()   for   p   in   model.parameters()   if
p.requires_grad)
    return {'Total': total_num, 'Trainable': trainable_num}
print(get_parameter_number(model))

# 定义日志（data 文件夹下，同级目录新建一个 data 文件夹）
def write_log(w):
    file_name  =  'data/'  +  datetime.date.today().strftime('%m%d')+"_{}.
log".format("deepfm")
    t0 = datetime.datetime.now().strftime('%H:%M:%S')
    info = "{} : {}".format(t0, w)
    print(info)
    with open(file_name, 'a') as f:
        f.write(info + '\n')
```

5. 训练与推断

```
def train_and_eval(model, train_loader, valid_loader, epochs, device):
    best_auc = 0.0
    for _ in range(epochs):
        """训练部分"""
        model.train()
        print("Current lr : {}".format(optimizer.state_dict()['param_groups']
[0]['lr']))
        write_log('Epoch: {}'.format(_ + 1))
        train_loss_sum = 0.0
        start_time = time.time()
```

```
        for idx, x in enumerate(train_loader):
            cate_fea, nume_fea, label = x[0], x[1], x[2]
            cate_fea, nume_fea, label = cate_fea.to(device), nume_fea.to
(device), label.float().to(device)
            pred = model(cate_fea, nume_fea).view(-1)
            loss = loss_fcn(pred, label)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            train_loss_sum += loss.cpu().item()
            if (idx+1) % 50 == 0 or (idx + 1) == len(train_loader):
                write_log("Epoch {:04d} | Step {:04d} / {} | Loss {:.4f} |
Time {:.4f}".format(
                        _+1, idx+1, len(train_loader), train_loss_sum/(idx+
1), time.time() - start_time))
        scheduler.step()
        """推断部分"""
        model.eval()
        with torch.no_grad():
            valid_labels, valid_preds = [], []
            for idx, x in tqdm(enumerate(valid_loader)):
                cate_fea, nume_fea, label = x[0], x[1], x[2]
                cate_fea, nume_fea = cate_fea.to(device), nume_fea.to
(device)
                pred = model(cate_fea, nume_fea).reshape(-1).data.cpu().
numpy().tolist()
                valid_preds.extend(pred)
                valid_labels.extend(label.cpu().numpy().tolist())
        cur_auc = roc_auc_score(valid_labels, valid_preds)
        if cur_auc > best_auc:
            best_auc = cur_auc
            torch.save(model.state_dict(), "data/deepfm_best.pth")
        write_log('Current AUC: %.6f, Best AUC: %.6f\n' % (cur_auc,
best_auc))
```

```
train_and_eval(model, train_loader, valid_loader, 30, device)
```

6. 结果展示

推荐结果展示如图 5-4 所示。

```
Current lr : 0.00034359738368000027
10:45:51 : Epoch: 13
10:45:54 : Epoch 0013 | Step 0050 / 196 | Loss 0.4403 | Time 3.3347
10:45:58 : Epoch 0013 | Step 0100 / 196 | Loss 0.4417 | Time 6.7146
10:46:01 : Epoch 0013 | Step 0150 / 196 | Loss 0.4450 | Time 10.1132
3it [00:00, 25.49it/s]
10:46:04 : Epoch 0013 | Step 0196 / 196 | Loss 0.4466 | Time 13.2456
25it [00:01, 22.85it/s]
10:46:05 : Current AUC: 0.780754, Best AUC: 0.780754

Current lr : 0.00027487790694400024
10:46:05 : Epoch: 14
10:46:09 : Epoch 0014 | Step 0050 / 196 | Loss 0.4259 | Time 3.3697
10:46:12 : Epoch 0014 | Step 0100 / 196 | Loss 0.4299 | Time 6.7538
10:46:16 : Epoch 0014 | Step 0150 / 196 | Loss 0.4331 | Time 10.1182
3it [00:00, 25.93it/s]
10:46:19 : Epoch 0014 | Step 0196 / 196 | Loss 0.4356 | Time 13.1798
25it [00:01, 22.81it/s]
10:46:20 : Current AUC: 0.777718, Best AUC: 0.780754
```

图 5-4　推荐结果展示

7. 项目实战

（1）项目背景：个性化体检项目推荐，根据用户画像和产品画像，推荐适合客户的个性化项目。

（2）医学规则：根据用户画像来推荐检查的项目，举例如下。

☑ 动脉硬化检查：必须进行常规检查。

☑ 重金属检查：需进行常规、血型检查。

☑ BV、HPV、TCT、TS 检查：做这些化验必须选择“妇科一般检查”。

☑ 男性检查：不能选择 CA125、CA153、妇科一般检查、TCT、BV、HPV、TS、盆腔超声、乳腺超声、乳腺钼靶、女性激素八项、女内外。

☑ 女性检查：不能选择总前列腺抗原、游离前列腺特异性抗原、前列腺超声、男性激素八项、精液常规、男内外。

☑ 18 岁以下：不能选择 X 双能射线骨密度、所有 DR、所有 CT、妇科一般检查、TCT、

BV、HPV、TS。

（3）数据集：对用户的历史加项数据进行脱敏处理后，进行数据集的治理和归一化，最后形成训练集、测试集和验证集。

5.4 模型训练代码实战

本节以 xDeepFM 推荐算法为例，通过代码介绍 xDeepFM 的训练过程。代码如下。

```
# -*- coding: utf-8 -*-
from sklearn.metrics import log_loss, roc_auc_score
from sklearn.model_selection import train_test_split
from deepctr_torch.models import *
from config.train_config import *
from common.train_model import *
from sklearn.preprocessing import LabelEncoder
from deepctr_torch.inputs import get_feature_names

key2index = {}
model_name = "xdeepfm"

#1.文件或者数据作为数据源
def train_xdeepfm(data_path=None, df_data=None):
#2.读取数据源
    df_data = get_dataframe(data_path, df_data)
#3.根据场景确定关键特征（考虑增加特征项，如历史检查中的异常结果、历史选择检查项）
    df_data = df_data.dropdrop(
        ["check_year", "check_month", "check_day", "check_dayofweek", "reverse_
year", "reverse_month",
         "reverse_day", "reverse_dayofweek", 'add_package_8', 'add_package_
9', 'add_package_10',
         'add_package_11', 'add_package_12', 'add_package_14', 'add_package_
13', "combo_id"], axis=1)
#4.关键特征转换为类别特征，变化为离散值，add_pack_own_id 是一个自费加项目（自费加项包之间有关联关系，如互斥等，因此建议改用加项包组合，如 123、134 等）
```

```
    sparse_features = ["exam_id", "language_id", "relation_id", "gender",
"marry_id", "branch_id",
                       "city_id",  "brand_id",  "report_authority",  "age",
"add_pack_own_id"]
#5.空特征数据，变化为-1
    df_data[sparse_features] = df_data[sparse_features].fillna('-1', )
    df_data["age"] = df_data["age"].astype("int")
    target = ['label']
    label_count = 0
#6.统计一下所有自费加项目被买了多少次
    for label in df_data["label"]:
        if label == 1:
            label_count += 1
#7.初始化一些关键特征
    # 1.Label Encoding for sparse features,and do simple Transformation
for dense features
    df_dict = dict({"Unnamed: 0": "unknown", "label": 0, "reschedule":
"unknown"})
    for feature_name in sparse_features:
        df_dict[feature_name] = -1
#8.多加一行，每一个特征都初始化为-1，目的是为了预测的时候使用，特殊处理不规范的数据，提
升泛化能力。
    df_data.loc[len(df_data)] = df_dict
    le_dict = dict()
    for feat in sparse_features:
        le = LabelEncoder()
#9.对训练集的每一个特征进行编码，然后放入一个字典 le_dict
        df_data[feat] = le.fit_transform(df_data[feat])
        le_dict[feat] = le

    # Notice : padding=`post`

    # 2.count #unique features for each sparse field,and record dense
feature field name
#10.将训练集的特征拼凑为 list 格式
    fixlen_feature_columns = [SparseFeat(feat, df_data[feat].nunique())
                              for feat in sparse_features]

#11.xdeepfm 网络输入特征需要两个 list
```

```
    dnn_feature_columns = fixlen_feature_columns
    linear_feature_columns = fixlen_feature_columns
#12.xdeepfm 网络输入特征需要两个 list
    feature_names = get_feature_names(
        linear_feature_columns + dnn_feature_columns)
#13.划分测试集 0.1，训练集 0.9
    # 3.generate input data for model
    train, test = train_test_split(df_data, test_size=test_size)

    train_model_input = {name: train[name] for name in feature_names}
    test_model_input = {name: test[name] for name in feature_names}

    # 4.Define Model,train,predict and evaluate
#14.训练的设备类
    device = get_device(use_cuda)
    model = xDeepFM(linear_feature_columns=linear_feature_columns, dnn_
feature_columns=dnn_feature_columns,cin_layer_size=(256,), cin_split_half=
True, cin_activation='relu',dnn_dropout=dropout, task='binary', l2_reg_
embedding=1e-5, device=device)
    model.compile("adagrad", "binary_crossentropy",
                  metrics=["binary_crossentropy", "auc"], )
#15.训练模型
    model.fit(train_model_input, train[target].values,batch_size=batch_ size,
epochs=epoch, validation_split=validation_size, verbose=verbose, use_double=
True)
#16.预测，通过 log_loss 计算损失误差
    pred_ans = model.predict(test_model_input, 256)
    print("tests LogLoss", round(log_loss(test[target].values, pred_ans),
4))
    print("tests AUC", round(roc_auc_score(test[target].values, pred_ans),
4))
#17.保存模型
    # save
    model_path, parameter_path = get_model_path(model_name, epoch)
    save_model(model, model_path)
    save_parameters_and_le(linear_feature_columns, le_dict, sparse_features,
parameter_path)

def column_split(x):
```

```
    """
    行处理
    :param x: str
    :return: list
    """
    key_ans = x.split(',')
    for key in key_ans:
        if key not in key2index:
            # Notice : input value 0 is a special "padding",so we do not
use 0 to encode valid feature for sequence
            # input
            key2index[key] = len(key2index) + 1
    return list(map(lambda y: key2index[y], key_ans))


if __name__ == '__main__':
    # example
    # file_path = r"D:\cp\data\test\df_data_self_pack_aug.csv"
    file_path = ""
train_xdeepfm(data_path=file_path)
```

xDEEpFM 算法训练完毕后，通过以下代码进行测试。

```
import pytest
from deepctr_torch.models import xDeepFM
from common.train_model import get_test_data, SAMPLE_SIZE, get_device,
check_model

@pytest.mark.parametrize(

'dnn_hidden_units,cin_layer_size,cin_split_half,cin_activation,sparse_fe
ature_num,dense_feature_dim',
    [((), (), True, 'linear', 1, 2),
     ((8,), (), True, 'linear', 1, 1),
     ((), (8,), True, 'linear', 2, 2),
     ((8,), (8,), False, 'relu', 2, 0)]
)
def   test_xdeepfm(dnn_hidden_units,   cin_layer_size,   cin_split_half,
cin_activation, sparse_feature_num,
               dense_feature_dim):
```

```
    """
    测试 xdeepfm 运行
    :param dnn_hidden_units: list,list of positive integer or empty list,
the layer number and units in each layer of
                          deep net
    :param cin_layer_size: list,list of positive integer or empty list,
the feature maps  in each hidden layer of
                          Compressed Interaction Network
    :param cin_split_half: bool.if set to True, half of the feature maps
in each hidden will connect to output unit
    :param cin_activation:  activation function used on feature maps
    :param sparse_feature_num: int
    :param dense_feature_dim:int
    """
    model_name = 'xDeepFM'

    sample_size = SAMPLE_SIZE
    x, y, feature_columns = get_test_data(sample_size, sparse_feature_
num=sparse_feature_num, dense_feature_num=sparse_feature_num)
    model = xDeepFM(feature_columns, feature_columns, dnn_hidden_units=
dnn_hidden_units, cin_layer_size=cin_layer_size,
                    cin_split_half=cin_split_half,
cin_activation=cin_activation, dnn_dropout=0.5, device=get_device())
    check_model(model, model_name, x, y)


if __name__ == '__main__':
    pytest.main(["-s", "test_xdeepfm.py"])
```