第5章

点击率预估算法

第4章中已详细介绍了基于协同过滤信号的推荐系统。然而，在实际的推荐场景中，数据本身的特性给推荐算法落地带来了新的困难。首先，实际场景下的数据规模非常大，如“百度糯米”App在全中国有数以万计的优质商家，如果给每个用户直接从所有商家中进行推荐，无疑是非常耗时的。时效性影响了用户的使用体验，进而影响到用户信任感、用户黏性；也影响到网络平台的收益。因此，快速降低数据规模成为推荐系统的核心诉求之一。

本章共分为8节，5.1节将讨论推荐算法的时效性问题，介绍了推荐系统的召回和排序过程；从5.2节开始，将讨论如何利用用户、产品和场景等特征进行推荐的问题，具体而言，5.2节将详细介绍推荐系统中的点击率预测问题；5.3节～5.6节将分别介绍经典的用于点击率预测的框架和模型；5.7节将介绍结合深度学习技术的点击率预测框架；5.8节给出本章小结。

5.1 推荐系统中的召回和排序过程

5.1.1 为什么需要召回和排序环节

对于互联网公司而言，个性化的推荐系统的意义在于通过服务来提高用户使用黏性，以期提高公司的收入；对于用户而言，推荐系统需要在互联网上爆炸的信息中精准地推荐用户感兴趣的信息，提高用户的使用体验。试想如下场景：用户面对海量的商品列表时无从下手，而电商平台推荐的产品却又不对用户的胃口——用户很可能草草看了几页产品便因为没看到合适的产品而退出；但如果平台此时正好推荐了用户心仪的商品，则将会引发用户的兴趣，帮助用户提升了浏览体验，提升用户的购买概率，同时满足了用户和商家的需求。因此，提升推荐模型的性能是一个亟待解决的现实问题。

一般而言，推荐系统的性能需要达到两方面的要求，分别是准确性和实时性两个标准。准确性指的是给用户推荐的产品契合了用户的兴趣(用户与被推荐的产品发生交互)；实时性指的是推荐系统的决策时间不宜过长。每一个用户推荐时，为了避免用户的满意程度下降，应满足实时推荐的需求。

为了提升准确性，推荐系统希望尽可能多地利用与推荐这一行为相关的“信息”。涉及用户的方面，指代的是用户的历史行为、性别、年龄等人口信息，以及社交网络关系等，可以用于建模用户的兴趣，被统称为“用户信息”；对于“物品”而言，在商品推荐中指的是商品的

种类、生产日期、价格等信息，在音乐推荐中指的是音乐风格、流行度等信息。简言之，可归纳为“物品信息”。同时，在特定的场景下，用户的选择会受到当前状态、地点、时间等因素的影响，统称为“上下文信息”或者“场景信息”。尽可能地收集相关信息是准确的个性化推荐列表的前提条件。相关的推荐系统示意图如图 5-1 所示。

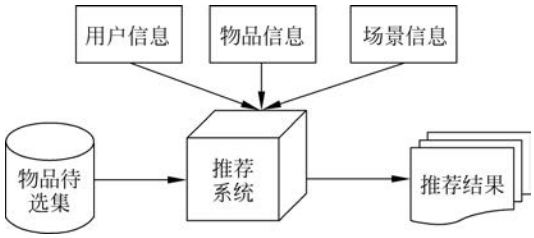

图 5-1　推荐系统示意图

然而，如图 5-1 所示的推荐系统示意图难以满足实时性。其原因在于，当前互联网平台候选物品集的数量巨大——百万级别，甚至千万级别的候选物品都在用户潜在的可能感兴趣的范围之内。计算每个用户对百万级别的产品的点击概率是非常耗时的。为此，工程师在设计模型时，对流程做了细化。具体而言，将流程细化分为“线下训练”“评估”和“线上推断”等阶段。“线下训练”的阶段是通过收集好的数据集，在线下对整个推荐系统模型的参数、权重等进行调整；“评估”阶段则是评判一个推荐系统是否可以达到投入实际使用的标准；“线上推断”则利用训练好的模型，给用户快速、准确地提供推荐服务。同时，为了调和规模巨大的候选物品集与“实时性”的冲突，工程师们设计了推荐系统模型中的“召回层”“排序层”，通过“召回层”减小候选物品集的规模。

如图 5-2 所示，“召回层”阶段的目的是从海量的物品库中寻找用户可能感兴趣的产品，其需求是快，并且不遗漏用户感兴趣的产品。为此，召回一般使用多种策略，主要是一些简

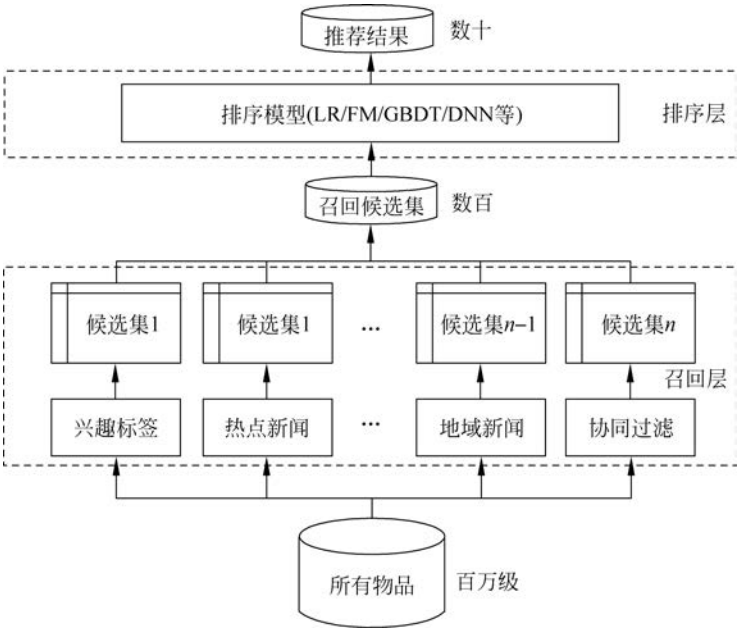

图 5-2　推荐系统的召回层、排序层

单的模型或者手工规则，避免复杂模型占用较多的时间。如用户的兴趣标签匹配模型、协同过滤表征匹配模型、根据产品热门度召回等。同时，“召回层”需要的特征通常是粗粒度的标签，具体包括用户的兴趣标签、产品的标签、热门度等即可，精细的特征会降低召回的速度。经过召回阶段，排序的产品集数量应当从百万级别快速降低到千级别或百级别，并且筛选出的都是用户有很大概率感兴趣的物品。

“排序层”的目的是为每个候选样本预测出精准的偏好分数，给出准确的推荐列表。“召回层”将候选集的规模降低到了百级别，大大减轻了计算压力，因此，“排序层”可以采用复杂的模型和更加细粒度的特征来支撑训练而不必担心时效性。

总之，“召回层”和“排序层”都是不可或缺的，“召回层”使得推荐系统满足了实时性的要求，“排序层”则解决了准确性的需要。总结其特点，“召回层”强调快，要求快速地缩小用户潜在感兴趣的范围，为下一步“排序层”减轻计算压力；“排序层”强调准，在已经缩小的用户感兴趣的范围内，精准地预测每一个候选样本的偏好分数，根据预测的偏好分数给出推荐的列表。

5.1.2 召回、排序环节的典型方法

5.1.1 节介绍了推荐系统模型中“召回层”和“排序层”的目标和特点。本节将延续对“召回层”和“排序层”的讨论，介绍在实际工程中“召回层”和“排序层”典型的实现方法。

首先，召回阶段的目的是快速地缩小用户潜在感兴趣的范围。因此，如果“召回层”召回的物品太多，那么则达不到实时性的要求；如果“召回层”召回的物品不准确，那么会导致用于排序的候选集不准确，进而影响推荐的质量。为了同时保证“召回层”的准确性和实时性，实际工程中通常选择用多种不同的策略去分别生成用户可能感兴趣的物品集，然后将这些不同的物品集作为“排序层”的候选物品集。如果直观地去理解，就是用户对物品感兴趣的原因是多种多样的。用户可能对热门的产品感兴趣，关注热点变化；用户也可能对一些符合个人兴趣偏好的产品感兴趣，如足球爱好者很可能对 2021 年欧洲杯的进球集锦感兴趣。在实际中，通过不同的策略分别生成用户可能感兴趣的物品集的方式被称作“多路召回”。典型的召回策略如表 5-1 所示。

表 5-1 典型召回策略

召回方式	内容
基于热门产品的召回	假设用户对热门产品的关注度比其他产品会更高些。将产品按照热门程度排列，召回最热门的产品进入候选集
基于内容的召回	根据用户画像及产品特征进行匹配和召回。根据用户的年龄、职业、地域等特征有针对性地召回相应定位的产品
基于协同过滤的召回	考虑到用户与用户、产品与产品间的相似性，根据用户和产品的历史交互记录进行召回，在无需额外特征的情况下也可以自动学习用户和产品表征。根据利用协同信号的不同，分为 UserCF，ItemCF 和利用高阶协同信号的 PersonalRank、SimRank 等
基于社交信息的召回	基于丰富的用户社交信息。为用户召回了社交邻居单击过的产品。如看一看中的“好友都在读”“XXX 都在看”等相关推荐
基于场景的召回	基于当前的场景特征对产品进行召回。如利用用户刚单击过的产品这一场景特征，推荐相类似的其他产品

通过“多路召回”的方式，可以获得一个合适的候选集，大大缩小了用户可能感兴趣的产品的范围。但是千级别或者百级别的产品的数量依然略大，不能直接呈现给用户。“排序层”在给定候选集的基础上，预测精准的偏好分数，给出推荐列表。排序层的模型是整个推荐系统算法中研究的核心，得到了学术界和工业界的广泛研究。提高排序层的结果准确度的方式一般有两种，分别是提高特征工程的水平和提高模型的能力。特征工程由领域内的专家手工设计，常见的实现形式有特征交叉等。排序层的经典模型也层出不穷，既包含第4章提到的经典的协同过滤(Collaborative Filtering，CF)模型[14]，也包含在本章的后续即将介绍的点击率预测模型，如逻辑回归(Logistic Regression，LR)模型[1]，因式分解机(Factorization Machine，FM)[2]、基于集成学习的梯度提升树(Gradient Boosting Decision Tree，GBDT)[8]，以及基于深度学习的Wide&Deep组合模型[5]框架等。5.2节将给出点击率预测的简介；并在5.3节～5.7节，聚焦点击率预测任务，分析不同模型的优缺点，理清不同模型之间的关系，展现点击率预测技术演变的过程。

5.2　点击率预测简介

如5.1节所述，点击率(Click Through Rate，CTR)预测模型是排序阶段的一类重要模型。点击率预测模型的任务就是根据用户的特点预测用户对商品点击的概率，将用户更可能点击的物品优先推荐给用户。点击率预测有着广泛的应用场景，如广告推荐、新闻推荐、歌曲推荐等。一个优秀的点击率预测模型，应当向用户展示其有可能感兴趣的产品，进而可以提升用户的使用体验，提升平台的收益。

为了保证准确性，CTR预测模型往往使用多种模态的特征信息来判断用户的偏好。其中，特征相互之间的差异很大，并且不同特征对于推荐结果的影响力也大相径庭。为了衡量特征对于点击率预测结果的不同影响，一般而言，点击率预测模型为每一种特征赋予一个权重，权重代表相对应的特征对点击率结果的影响力大小。通过最小化预测概率和真实点击行为之间的差异，来训练模型，调整每个权重的大小，最终训练得到多模态的特征信息到是否点击的映射函数。

经过研究者的分析发现，仅利用原始的特征，点击率预测模型的性能提升有限。幸运的是，用户的历史行为之中包含许多隐藏的特征组合关系，而这些特征组合关系的挖掘能够直接有效地提高点击率预测的准确度。常见的特征组合关系举例如下。

(1) 用户经常在中午11点左右，打开“百度糯米”软件，其中，中午11点是一个时间信息，一般而言，这是准备吃午饭的时间；App是“百度糯米”，该信号便是一个时间和App的组合，这样的信号被视作一个二阶特征组合关系。

(2) 某男性用户经常爱听周杰伦的歌又爱玩模拟对抗游戏，在这个案例中，用户的性别、歌曲偏好、游戏偏好组成一个信号，这个信号是一个三阶特征组合关系。为了便于表达，用AND符号表示这种关系。如上例可以表达为AND(性别男，爱听周杰伦的歌，爱玩游戏)，对于这个样本，这个三阶交叉特征为真。

广义线性模型没有自动学习特征组合关系的能力。因此，实际中常用的方法是请专家进行特征工程，然后将特征工程产生的新特征和原特征一起赋予权重，参与点击率预测模型的训练。

通过特征工程的方式挖掘特征组合极大地提高了点击率预测模型的准确度。然而，特征工程依然有着明显的缺点，例如，依赖于场景和任务，切换场景和任务需要进行重新特征工程；需要耗费大量人力物力等。如图 5.3 所示，为了弥补特征工程的缺陷，提升点击率预测模型的性能，Rendle 等人提出因式分解机（Factorization Machine，FM）[2]，FM 将特征之间的二阶组合关系矩阵分解，为每一个特征分配一个低维的隐式向量。某个二阶特征组合关系的权重等于该组合关系对应特征的隐式向量内积，该方法可在数据稀疏的数据取得更好的效果。FM 在理论上可以扩展到高阶，但由于模型复杂度太高，实际应用只使用二阶 FM。为了避免特征交叉的维度变高，导致组合爆炸、复杂度高等问题，Facebook 提出了 GBDT＋LR[4]，利用 GBDT 对特征自动进行筛选和组合，进而生成新的离散的特征变量；随着深度学习的崛起，研究者们开始探索深度学习对于 CTR 预测任务的优势。Google 推荐算法团队，创新性地合二为一，综合利用浅层和深度模型，至此，Wide&Deep[5] 横空出世，进行联合训练，在当时的工业界和学术界引起了巨大的反响，时至今日依然是业界的主流模型之一。5.3 节～5.7 节的内容，将围绕图 5-3 展开。

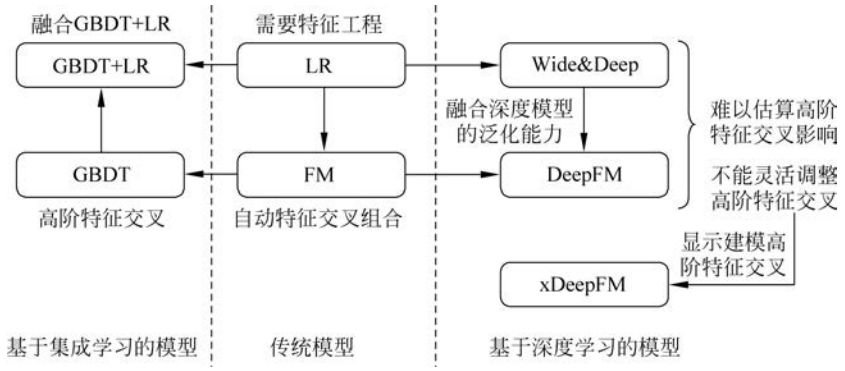

图 5-3　CTR 模型的演变

图 5-3 简单介绍了本章介绍的 CTR 模型之间的演变关系。可以看到，逻辑回归（LR）模型是一种最基础的 CTR 预测的模型。5.3 节将具体介绍 LR 模型原理，并总结其优缺点。

5.3　逻辑回归模型

5.3.1　背景

为了有效地优化推荐结果，直观的想法是将“用户信息”“物品信息”和“上下文信息”等所有的信息融合起来送入一个机器学习模型中，学习一个特征信息到点击率预测概率的函数。

显然，每种特征对于最终预测的点击率的影响力大相径庭。为了衡量不同特征差异化的影响力，回归模型是一个简单直接的选择。回归模型可以给每种特征赋予一个权重，通过调整预测概率和实际单击与否的标签来调整权重的大小，以此修改每种特征对点击率预测结果的影响力。

在实际使用中，因为实际单击的标签分为单击和不单击（1 和 0），点击率预测问题也可以被看作一个二分类问题。因此，研究者们使用逻辑回归（Logistic Regression，LR[1]）模型来代替普通的线性回归：通过引入 Sigmoid 层，将输出的结果限定在（0，1）的区间内；利用

Log-loss 来代替最小二乘的优化目标，如图 5-4 所示。

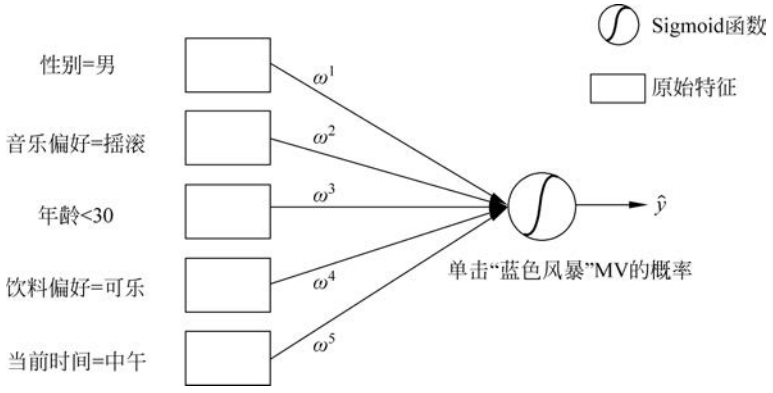

图 5-4　原始特征到单击概率的映射关系

研究者们发现，引入交叉特征组合对 CTR 预测的效果提升有直接的影响。原始的 LR 模型很难从原始特征输入中捕捉特征交叉的影响。5.2 节中举例说明了二阶三阶交叉特征组合的例子。为了直观地体现交叉特征对模型的影响，图 5-5 重画了逻辑回归的输入。具体而言，将特征进行交叉组合，然后将特征组合和原始特征一起作为逻辑回归模型的输入。

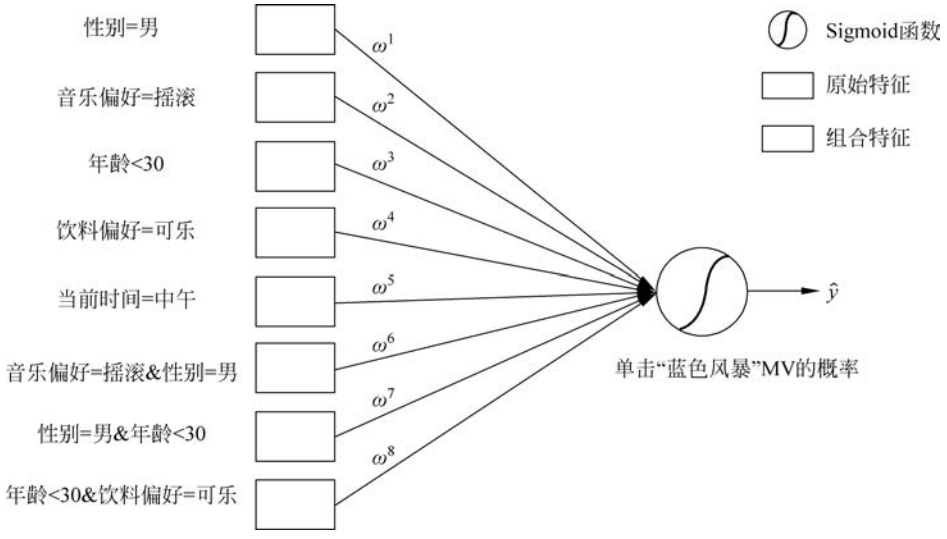

图 5-5　特征工程后，特征到单击概率的映射关系

具体的逻辑回归模型的流程将在 5.3.2 节介绍。

5.3.2　基于 LR 模型的 CTR 预测流程

将逻辑回归模型应用于点击率预测任务，一般要经历如下步骤：①将用户的连续型特征，如年龄、性别、当前时间地点等信息变换为数值型的特征向量；②对原始数值型特征向量进行特征工程，构建相关的交叉特征组合；③在模型训练阶段，以提高点击率预测的准确程度为目标，利用已有样本数据（包含原始特征、交叉特征和是否单击的结果）对逻辑回归模型进行训练，调整模型参数；④在面向用户的模型服务阶段，将当前的特征信息输入 LR 模型，模型正向传播输出预测结果，根据推断的结果得到用户单击产品的概率；根据概率大小

进行排序，生成最终的推荐列表。

其中，重点介绍交叉特征、训练模型等过程。

1. 交叉特征的构造过程

“常见的特征组合关系举例如下：①用户经常在中午 11 点左右，打开‘百度糯米’软件，其中，中午 11 点是一个时间信息，一般而言，这是准备吃午饭的时间；App 是‘百度糯米’，该信号便是一个时间和 App 的组合，这样的信号被视作一个二阶特征组合关系；②某男性用户经常爱听周杰伦的歌又爱玩模拟对抗游戏，在这个案例中，用户的性别、歌曲偏好、游戏偏好组成一个信号，这个信号是一个三阶特征组合关系。”

如上所述，5.2 节已经给出过二阶以及三阶的特征交叉的例子，并介绍了这样的交叉特征能够直接影响到点击率预测的准确性，但需要专家的领域知识。在这里，提供一种直观的理解，为什么交叉特征对于点击率预测的准确性有利以及为何说需要专家的领域知识？

首先以二阶交叉特征举例。假设某用户经常在 11 点打开“百度糯米”App 选择午饭，倾向于在 14 点打开“百度”App 搜索资料，在 20 点经常打开“百度贴吧”搜索感兴趣的内容。以上都是时间特征和 App 名称的二阶特征。如果将这些二阶特征拆开还原成一阶信号特征，其形式应该是①用户倾向于在 11 点、14 点、20 点使用手机；②用户倾向于使用“百度”“百度贴吧”“百度糯米”这些软件。

如果用户在一天内浏览“百度糯米”App 的次数最多，那么只接收一阶信号的 LR 模型很有可能在晚上 8 点也认为用户最有可能打开“百度糯米”。这样的原因可能是用户单击“百度糯米”的次数相当多，以至于 LR 模型在更新权重的时候，单击“百度糯米”对应的权重被调整得非常大。因此，在晚上 8 点，推荐系统也认为用户最有可能打开“百度糯米”。这样的推荐结果显然是不利的。对大多数人来说，晚上 8 点可能刚吃完晚饭不久，推荐他们打开一个外卖 App 是略显荒谬的，他们此刻可能并不想享受美食。归根结底，之所以出现这样的情况，是因为大部分时候用户的单一特征与其行为没有特别大的关联。

相反地，多个特征交叉的组合往往可以解释用户行为，这是基于对真实现象的观察得出的规律。这些交叉特征组合往往需要拥有丰富经验的领域内的专家设计得出。现实生活中，还有非常多的情况满足用户的行为组合特征密切相关的实例，例如：①“青年人”这一特征和“购买甜筒”这个行为没有什么特别大的关联，但是兼具“性别为女”且“青年人”且“带着小孩”这个三阶交叉特征组合的人则有很大概率进行“购买甜筒”这个行为，很可能因为具备这个三阶交叉特征组合的人是为了安抚小孩特意去购买了甜筒；②“性别为男”和“购买啤酒”这个行为可能有一定的关联，但“性别为男”且“当前处于夏天”这个二阶交叉特征组合则有更大的概率导致“购买啤酒”这个行为。显然，构建特征交叉需要对当前推荐场景有丰富的领域知识。换言之，只有请领域内的专家进行设计，才能有效构建出对点击率预测有利的交叉特征。

2. LR 模型的训练过程

如图 5-5 所示，将特征工程后的组合特征和原始特征拼接在一起，并赋予每一个特征一个独立的权重参数。公式表达如下。

$$\hat{z}=\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}=\omega_0 x_0+\omega_1 x_1+\cdots\omega_n x_n=\sum_{i=0}^{n}\omega_i x_i \tag{5.1}$$

其中，$\boldsymbol{x}$ 即为模型输入的各个特征（包含原始特征和特征工程的新特征集），$\boldsymbol{\omega}$ 即为模型需要学习的参数，代表每种特征对预测结果影响的权重，$\hat{z}$ 代表根据当前的权重参数和样本所估

算出的隐变量。不同于普通线性回归的是，逻辑回归在 Sigmoid 函数的基础上引入了非线性因素，将输出的范围缩放到了(0,1)这个区间内，符合“点击率”这个概念的物理意义。

从而得到了参数值到最终预测结果的映射，表示如下。

$$h_{\omega}(\boldsymbol{x})=\frac{1}{1+\mathrm{e}^{-\hat{z}}}=\frac{1}{1+\mathrm{e}^{-\sum_{i=0}^{n}\omega_i x_i}} \tag{5.2}$$

通过以下的 Log-loss，借助梯度回传，通过调整$\boldsymbol{\omega}$的取值，减小预测结果到真实标签的差异。

$$J(\boldsymbol{\omega})=-\frac{1}{n}\left(\sum_{i=0}^{n}\left(y^i\log h_{\omega}(x^i)+(1-y^i)\log(1-h_{\omega}(x^i))\right)\right) \tag{5.3}$$

值得一提的是，后续介绍的 CTR 预测任务基本都采取相同的损失函数形式，后面不再赘述。

LR 模型的线上推断过程并不复杂。在模型上线后，通过当前输入的特征 x，固定模型训练的权重，通过预测候选物品集合的预测概率，根据预测概率对候选物品排序，给出最终的推荐列表。

LR 模型在很长一段时间内都是推荐系统、广告投放领域的主流模型之一。因为其突出的优点——模型简单、可解释性强、易于并行、满足工程化的需要，在工业界得到了应用。但是，LR 模型的缺点也很明显，它自身没有学习特征组合关系的能力。为了学习到数据中的特征组合关系，常用的方式是请专家进行特征工程，将特征工程获得的特征填充到原始特征中。这种方式有着明显的弊端：①依赖于场景或任务，每当点击率预测的场景迁移，就需要一个深入理解新场景中业务逻辑的专家来重新设计特征工程，需要巨大的人力成本；②网络环境下包含海量数据，用户行为背后的特征组合关系更加复杂；因此，即使耗费巨大的人力成本，人工进行特征工程也十分困难。

5.3.3　实验

本章实验部分大体上将依托于百度公司官方提供的 PaddleRec 推荐代码[①]展开，在 Python 3 的环境下安装 PaddlePaddle[②] 和 scikit-learn[③] 等机器学习包。

1. 数据准备

Criteo 数据集是一个经典的点击率预估数据集，其目的是通过 Criteo 公司的一周时间跨度的部分真实流量数据去预测广告被点击的概率。具体而言，该数据包含 13 种数值型特征和 26 种类别型特征。我们下载了 Criteo 的全数据集[④]，包括约 4000 万的训练集数据和约 600 万的测试集数据。我们使用训练集里的数据来训练模型，优化模型的参数使模型输出拟合训练集数据；用测试集上的数据来计算误差，并以此估计模型在应对真实场景中的泛化误差。

2. 主要代码

通过引入截取 Wide&Deep 模型中的 Wide 部分，获得了最简单的 LR 模型格式。具体而言，LR 模型部分的核心仅仅是在模型的初始化函数(__init__)中定义了一个线性层函数

① https://github.com/PaddlePaddle/PaddleRec/

② https://www.paddlepaddle.org.cn/

③ https://scikit-learn.org/stable/

④ Criteo 全训练集下载地址：https://paddlerec.bj.bcebos.com/datasets/criteo/slot_train_data_full.tar.gz
Criteo 全测试集下载地址：https://paddlerec.bj.bcebos.com/datasets/criteo/slot_test_data_full.tar.gz

paddle.nn.Linear。该函数输入的维度是特征维度，输出的维度是1，代表预测的概率。

在模型的训练阶段，将会执行 forward 函数，每个 batch 的数据将先通过线性层缩放到一维，再经过 Sigmoid 层映射到 0～1 范围。forward 函数输出每个 batch 的预测结果(pred)，该结果可以和真实标签构成 loss 函数，通过优化器优化 LR 模型的参数。代码如图 5-6 所示。

```
import paddle
import paddle.nn as nn
import paddle.nn.functional as F
import math
class LR(nn.Layer):
    def __init__(self, sparse_feature_number, sparse_feature_dim,
                 dense_feature_dim, num_field, layer_sizes):
        super(LR, self).__init__()
        self.sparse_feature_number = sparse_feature_number
        self.sparse_feature_dim = sparse_feature_dim
        self.dense_feature_dim = dense_feature_dim
        self.num_field = num_field
        self.layer_sizes = layer_sizes
        self.wide_part = paddle.nn.Linear(
            in_features=self.dense_feature_dim,
            out_features=1,
            weight_attr=paddle.ParamAttr(
                initializer=paddle.nn.initializer.TruncatedNormal(
                    mean=0.0, std=1.0 / math.sqrt(self.dense_feature_dim))))
    def forward(self, dense_inputs):
        wide_output = self.wide_part(dense_inputs)
        pred = F.sigmoid(wide_output)
        return pred
```

图 5-6　LR 模型代码

3. 实验结果

本章实验的评价指标采用了 AUC(Area Under Curve)。具体而言，AUC 被定义为 ROC 曲线和 x 坐标轴所围区域的面积，其取值范围为 0.5～1。AUC 越高，则代表模型预测得越准确，也就是模型的性能越好；反之，AUC 越接近 0.5，越说明预测结果不准确。

在本次的 LR 实验的模型性能评估部分，在测试集上取得了 AUC 为 0.67 的结果。这样的结果虽然有一定准确度，但总体上不算优秀的结果。这主要是因为 LR 模型不能学习到特征之间的交叉，限制了实验性能的提升。

5.4　因式分解机模型

5.4.1　背景

5.3 节介绍了 LR 模型，也提到了其明显的缺点：对特征工程的依赖性。为了解决模型的缺点，摆脱烦琐的人工标注特征的方式，研究者们提出自动交叉特征模型的设想。Lin 等人提出了 Poly-2 模型[15]，它为每一个可能存在的特征组合关系都分配了一个独立的权重。虽然该模型解决了需要特征工程的问题，但是该模型无法学习到历史数据中很少或者没有出现过的组合关系的权重，这在高维稀疏数据的情况下非常普遍。例如，推荐系统根据所在地点(精确到地级市)、年龄段的二阶交叉特征来推荐产品。在这个例子里，地级市和年龄段的特征维度都相对稀疏，在历史记录中，很多交叉特征只有寥寥的单击数据，如只有很少的 A 市的 35～40 岁的用户的历史单击记录。因为数据的缺乏，很难为“A 市的 35～40 岁的用

户”这样的交叉特征学习到一个可靠的权重。

为了解决 Poly-2 模型带来的高维稀疏数据中特征权重无法学习的问题，2010 年，Rendle 等人创新性地提出了一种基于矩阵分解的机器学习算法——因式分解机(Factorization Machine，FM[2])，核心在于为每一个特征分配一个低维的隐式向量(Latent Vector)，二阶特征组合关系的权重等于该组合关系中的特征的对应的隐式向量内积。在高维稀疏的特征向量上，该模型展现了优秀的性能。总之，FM 模型成功地摆脱了特征工程，并且在稀疏的特征交叉数据上的权重学习上表现出了相对不错的效果。

5.4.2　FM 模型原理

如果直接在 LR 模型的基础上，引入所有二阶线性特征的估算，则点击率预估的模型表达如下。

$$\hat{y} = b + \sum_{i=0}^{n} \omega_i x_i + \sum_{i=0}^{n} \sum_{j=i+1}^{n} \omega_{ij} x_i x_j \tag{5.4}$$

其中，$\hat{y}$ 代表预测的单击概率，b 代表网络训练的偏置值，n 是样本特征的数量，ω_{ij} 代表二阶交叉特征对应的权重，x_i 代表第 i 个特征的取值。显然，这样的方式至少有两个明显的缺点。

(1) 在这样的计算方式下，二阶的特征权重参数的数量是 n^2 数量级。

(2) 只有当 x_i，x_j 都不是 0 时，对应的权重参数才有意义。如果满足都不是 0 这一条件的样本数量很少，则 ω_{ij} 对应的二阶交叉特征的数据非常稀疏，导致权重参数 ω_{ij} 难以得到有效的训练。

FM 则改进了这种方法。FM 引入了隐向量(维度为 k)来表示每一种特征，$\boldsymbol{v}_i$，$\boldsymbol{v}_j$ 分别代表特征 x_i，x_j 对应的隐向量。x_i，x_j 对应的交叉特征的权重则由 $\boldsymbol{v}_i$，$\boldsymbol{v}_j$ 的内积决定。由式(5.4)可知，每两种特征之间都对应一个权重参数，如图 5-7(a)所示。FM 的创新之处在于将图 5-7(a)的矩阵拆分成 $n \times k$ 的隐变量矩阵，两两特征之间的权重参数由对应特征的内积来表示，如图 5-7(b)所示。以此，将原本的 n^2 数量级的权重参数数量减少到了 nk 级别，大大降低了训练开销。同时，这种方法也使得数据稀疏的二阶交叉特征的权重参数可以得到较好的训练。

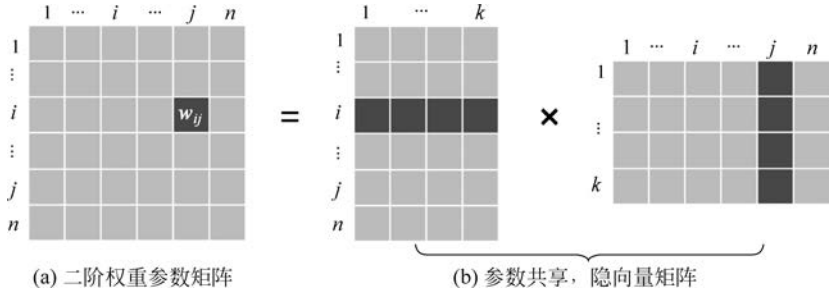

图 5-7　二阶权重参数矩阵分解

图 5-7 刻画了 FM 得名的由来，形象地解释了“分解”(Factorization)的含义。对应地，式(5.4)应当按照 FM 的方式进行调整，表达如下：

$$\hat{y} = b + \sum_{i=0}^{n} \omega_i x_i + \sum_{i=0}^{n} \sum_{j=i+1}^{n} < \boldsymbol{v}_i, \boldsymbol{v}_j > x_i x_j \tag{5.5}$$

其中，<，>代表两个向量的内积。式(5.5)描述了特征 x_i，x_j 及其对应的隐变量 $\boldsymbol{v}_i$，$\boldsymbol{v}_j$ 映

射到预测的点击率的过程。该模型的线上推断过程与 LR 类似，都是利用学习好的权重参数直接得出点击率的预测。

相比于简单的 LR 模型，FM 模型的优势非常明显。它避免了特征工程，可以自动地学习到二阶特征交叉的权重参数，并且模型的复杂度不算太高。在后续工作中，FFM[9] 和 DeepFM[6] 都在 FM 的基础上进行了扩展。但是对于二阶以上的特征交叉组合，单纯的 FM 模型在实际中会不可避免地导致组合爆炸和计算复杂度过高的问题。换言之，FM 在实际中不被用于自动建模三阶及以上的特征交叉。

5.4.3 实验

1. 数据准备

依托于 PaddleRec 的官方代码和 5.3 节相同的数据，使用 Criteo 全数据集。

2. 主要代码

如图 5-8 所示 FM 模型结构，定义了维度为 1 的 embedding 模块和 create_parameter 模块(参数矩阵)，并将 embedding 模块取出用来计算式(5.5)中的 $\sum_{i=0}^{n}\omega_i x_i$ 项，其中，执行 self.embedding(index)是取出索引为 index 的参数内容；self.multiply 则是通过乘法取得对应的参数内容。该项的结果对应 forward 函数中输出的 y_first_order 变量。

```
class FM(nn.Layer):
    def __init__(self, sparse_feature_number, sparse_feature_dim,dense_feature_dim, sparse_num_field):
        super(FM, self).__init__()
        self.sparse_feature_number = sparse_feature_number
        self.sparse_feature_dim = sparse_feature_dim
        self.dense_feature_dim = dense_feature_dim
        self.dense_emb_dim = self.sparse_feature_dim
        self.sparse_num_field = sparse_num_field
        self.init_value_ = 0.1
        #sparse part coding
        self.embedding_one = paddle.nn.Embedding(sparse_feature_number,1,sparse=True,
            weight_attr=paddle.ParamAttr(initializer=paddle.nn.initializer.TruncatedNormal(
                    mean=0.0, std=self.init_value_ / math.sqrt(float(self.sparse_feature_dim)))))
        self.embedding = paddle.nn.Embedding(
            self.sparse_feature_number, self.sparse_feature_dim, sparse=True,
            weight_attr=paddle.ParamAttr(initializer=paddle.nn.initializer.TruncatedNormal(
                    mean=0.0, std=self.init_value_ / math.sqrt(float(self.sparse_feature_dim)))))
        #dense part coding
        self.dense_w_one = paddle.create_parameter(shape=[self.dense_feature_dim],
            dtype='float32', default_initializer=paddle.nn.initializer.Constant(value=1.0))
        self.dense_w = paddle.create_parameter(shape=[1, self.dense_feature_dim, self.dense_emb_dim],
            dtype='float32',default_initializer=paddle.nn.initializer.Constant(value=1.0))
    def forward(self, sparse_inputs, dense_inputs):
        #---------------------- first order term ----------------
        sparse_inputs_concat = paddle.concat(sparse_inputs, axis=1)
        sparse_emb_one = self.embedding_one(sparse_inputs_concat)
        dense_emb_one = paddle.multiply(dense_inputs, self.dense_w_one)
        dense_emb_one = paddle.unsqueeze(dense_emb_one, axis=2)
        y_first_order = paddle.sum(sparse_emb_one, 1) + paddle.sum(dense_emb_one, 1)
        # ---------------------- second order term------------------
        sparse_embeddings = self.embedding(sparse_inputs_concat)
        dense_inputs_re = paddle.unsqueeze(dense_inputs, axis=2)
        dense_embeddings = paddle.multiply(dense_inputs_re, self.dense_w)
        feat_embeddings = paddle.concat([sparse_embeddings, dense_embeddings],1)
        #sum_square part
        summed_features_emb = paddle.sum(feat_embeddings,1)  #None*embedding_size
        summed_features_emb_square = paddle.square(summed_features_emb)  #None*embedding_size
        #square_sum part
        squared_features_emb = paddle.square(feat_embeddings)  # None *num_field*embedding_size
        squared_sum_features_emb = paddle.sum(squared_features_emb, 1)  #None*embedding_size
        y_second_order = 0.5 * paddle.sum(summed_features_emb_square - squared_sum_features_emb,1,keepdim=True)  #None
        return y_first_order, y_second_order
```

图 5-8 FM 模型结构

为了计算 $\sum_{i=0}^{n}\sum_{j=i+1}^{n}<\boldsymbol{v}_i,\boldsymbol{v}_j>x_ix_j$，定义了与上文类似的 embedding 模块和参数矩阵，其维度是 $\boldsymbol{D}$，对应着公式中的 $\boldsymbol{V}$ 隐向量矩阵。通过计算隐向量的内积，获得了二阶项的权重。该项的结果对应 forward 函数中输出的 y_second_order 变量。

3. 实验结果

FM 模型自动进行了二阶交叉特征，在这个微型数据集的测试集上，取得了 AUC 为 0.78 的模型性能。该结果高于 LR 模型的结果，原因是 FM 模型可以自动学习二阶交叉特征，提高了 CTR 预测的性能。

5.5　梯度提升树模型

为了弥补 FM 模型无法在实际中建模二阶以上的特征交叉的缺点，如何突破二阶特征交叉的限制，提升特征交叉的维度，是推荐系统研究者们努力的方向。本节将介绍一种基于集成学习的方法，一定程度上缓解了高阶特征组合的难题。

5.5.1　背景

梯度提升树(Gradient Boosting Decision Tree，GBDT[3][8])是应用最为广泛的机器学习的集成模型之一。集成模型的思想是将若干个基础模型或者弱学习器按一定的策略结合到一起，从而得到一个能力更强的集成分类器。集成学习的思想在中国谚语中有着直观的解释：这便是“人多力量大”和“三个臭皮匠，赛过诸葛亮”。一般而言，Boosting 采用串行的方式按一定的先后顺序训练各个学习器，通过前面学习器训练的经验来改善后续学习器的训练。具体做法是使用赋予训练集中的样本不同的关注度。在每次的迭代训练中，对于那些训练得已经很准确的样本，则给予更少的关注度；训练不好的权重，给予下一个弱学习器更高的关注度。当弱学习器 1 得到错误的结果时，下一个弱学习器将给予第一个学习器学错的样本更高的关注度。按照这样的方式，GBDT 最终能综合所有弱学习器的优势，形成一个强分类器。

在正式介绍 GBDT 算法之前，还需要介绍一般情况下 GBDT 的基学习器的结构——决策树(Cart Tree)。决策树模型属于二叉树。在训练过程中，决策树的每个节点按照某种规则进行树分裂，找到一个参考特征和分割点，将当前的样本集合拆分开。决策树模型的中止条件一般是预设的，如决策树达到某深度，或者叶子节点的数量达到要求，此时，决策树模型的树分裂阶段中止，新的样本可以根据树的结构预测输出值。每个叶子节点对应一个输出值，如果样本数量超过一个，以所有样本标签的平均值作为输出。在预测过程中，从根节点开始，对样本的特征进行测试，根据测试结果，将样本逐层进行分配，最终到达一个叶子节点，该叶子节点对应一个输出值。决策树模型的特点是简单、低方差、高偏差，随着 GBDT 级联的决策树越多，偏差逐渐变小，最终的模型可以同时满足较低的方差和偏差的要求。

5.5.2　模型原理

GBDT 通过逐一生成决策树的方式生成整个树林，生成新的决策树使用的标签是利用样本的真实标签值和当前的树林预测值之间的残差。当前的树林预测值由当前树林内所有

的树结构输出的总和决定。同时，因为残差是连续变量，因此 GBDT 的决策树一般采取回归树的结构。在 GBDT 中添加一棵新的回归树的过程如图 5-9 所示。

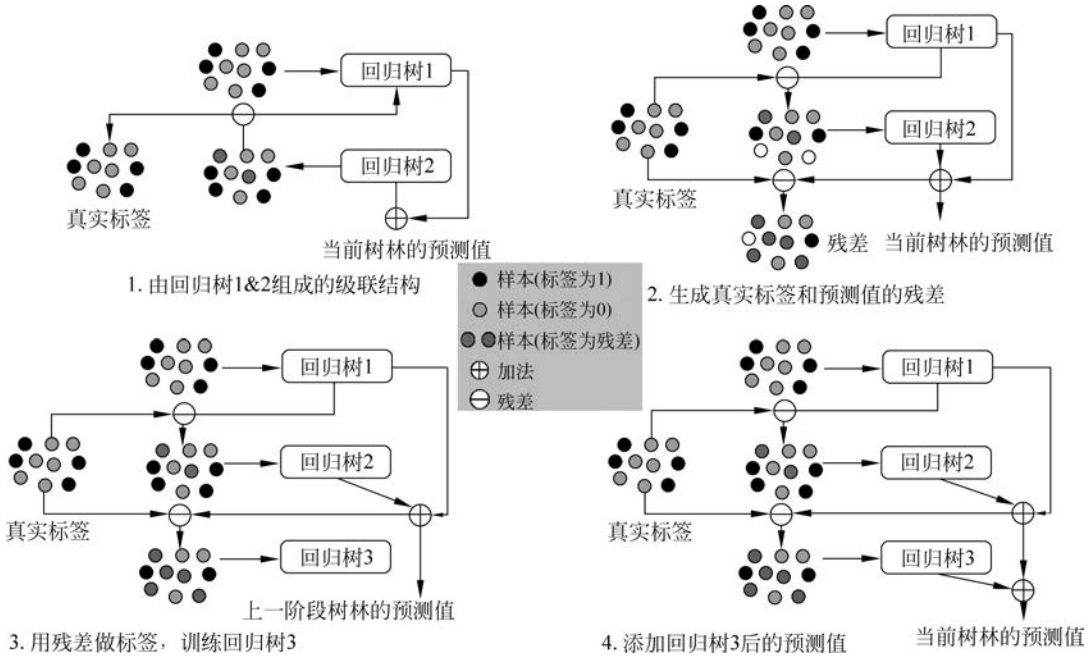

图 5-9 GBDT 中添加一棵新的回归树

每添加一棵新的回归树，首先要求和当前所有树林的预测结果，然后与真实标签做残差，并作为下一棵回归树的标签。假设当前已训练完成回归树组合 $T_1 \cdots T_M$，输入数据为 $[x_1, x_2, \cdots, x_n]$，则当前树林的预测结果可以表达为：

$$f_M(x) = \sum_{m=1}^{M} T_m(x) \tag{5.6}$$

如果添加了第 $M+1$ 棵回归树，则新的模型预测结果是：

$$f_{M+1}(x) = f_M(x) + T_{M+1}(x) \tag{5.7}$$

必须指出，梯度提升算法是利用最速下降的近似方法，即利用损失函数的负梯度在当前模型的值，作为回归问题中提升树算法的残差的近似值，拟合一棵回归树。平方差函数 $L_m(y, f(x)) = \frac{1}{2} \times (y - f_m(x))^2$ 被选择为目标函数，用平方误差最小的准则求解每个单元上的最优输出值。其梯度与残差的形式正好相契合。值得一提的是，本节的重点在于展示 GBDT 的流程，而不是探索 GBDT 算法关于不同损失函数的选择。因此，在本节的说明中，只说明第 $M+1$ 棵子树的拟合残差 $y - f_M(x)$ 的情况，而不考虑其他选择。

至此，如何在 GBDT 算法中逐次添加回归树模型已经清楚。后续内容将继续讨论基本学习器——回归树是如何工作的，以便于理解 GBDT 的内容，也为 5.6 节讨论的 GBDT+LR 做铺垫。

要理解回归树模型，首先要把思维从反向传播算法中解放出来。回归树模型参数学习的过程就是树分裂的过程，当所有分裂的分割向量和分割点都已经决定，单棵回归树此时已经训练完毕。如上文所述，残差就是平方差目标函数的负梯度值，训练新的回归树则以残差

作为标签，重新进行树分裂的过程。在实际应用中，单棵回归树的终止条件一般由指定的叶子节点数或者回归树的深度决定。

回归树的每次树分裂过程都会将当前节点分为两个子节点，其分类公式如下。

$$\min_{j,s}\left[\min_{c_1}\sum_{x_i\in R_1(j,s)}(y_i-c_1)^2+\min_{c_2}\sum_{x_i\in R_2(j,s)}(y_i-c_2)^2\right] \tag{5.8}$$

其中，y_i 是样本 i 的标签值，x^j 代表输入的第 j 种特征，s 代表切分点，R_1，R_2 分别代表被 x^j 和 s 区分开来的两个空间。例如，x^j 是用户年龄这一特征，切分点在 30 岁，则 R_1，R_2 可以是大于 30 岁和不大于 30 岁。式(5.8)的目标是，找到这样一组 x^j 和 s，能够使得分割开的两个空间满足有一组 c_1 和 c_2，能够使得目标函数最小。以此类推，直到达到回归树终止条件为止。

GBDT 的每棵回归树进行学习的过程中，都会经过数次交叉，每次交叉都以某个特征的某个值作为切分点。这样，在每棵回归树的学习过程中，GBDT 都会利用到高阶的交叉特征组合。

如图 5-10 所示，该回归树模型将输入的样本集合按特征分为四种，利用了 AND(“高薪”，“女性”)等二阶特征交叉。如果有更深的回归树模型，那么可以利用到的特征交叉的维度可以更高。用 GBDT 将多个不同回归树集成起来，每个回归树模型都对应着不同的高阶特征交叉，有效弥补了 FM 在引入高阶特征交叉时组合爆炸的问题。然而，只有 GBDT 的缺点也很明显，无法像 LR 一样，用权重参数建模特征对点击率的不同的影响力。为了弥补这个缺陷，相关研究者们继续开始了探索的旅程。

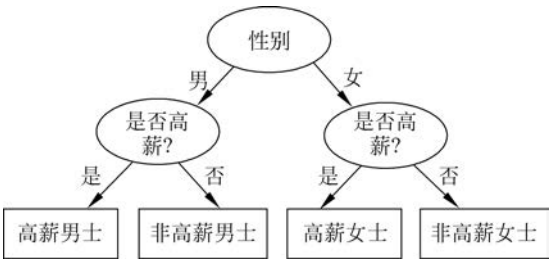

图 5-10　回归树示例

5.5.3　实验

1. 数据准备

因为树模型需要全部数据来计算节点分割的位置，而 Criteo 全部的训练集有 4000 万条数据，如果用 GBDT 树模型来对所有数据进行训练，需要的空间开销极大。因此，在本节(梯度提升树模型)和 5.6 节(梯度提升树＋逻辑回归模型)的实验中，我们替代性地使用了 Criteo 部分数据，约 1 万条数据，并按照 8∶2 原则拆分为训练集和测试集，分别用来训练模型和测试模型的性能。

2. 主要代码

我们在同样的微型数据集上，重写了数据读取函数，借助 lightgbm 构建树模型和 scikit-learn 的工具训练了 GBDT 模型，测试了其性能。如图 5-11 所示，我们从训练文件目录和测试文件目录每一行读取信息，并以 pandas 库的 DataFrame 形式保存。

```
def load_data(continuous_feature,category_feature):
    train_path = "./gbdt_data/sample_data/train/sample_train.txt"
    train_dict = defaultdict(list)
    with open(train_path, "r") as rf:
        for index,l in enumerate(rf):
            line = l.strip().split(" ")
            for i in line:
                slot_feasign = i.split(":")
                train_dict[index].append(float(slot_feasign[1]))
    train_pd = pd.DataFrame(train_dict).T

    test_path = "./gbdt_data/sample_data/test/sample_test.txt"
    test_dict = defaultdict(list)
    with open(test_path, "r") as rf:
        for index,l in enumerate(rf):
            line = l.strip().split(" ")
            for i in line:
                slot_feasign = i.split(":")
                test_dict[index].append(float(slot_feasign[1]))
    test_pd = pd.DataFrame(test_dict).T
    data = pd.concat([train_pd, test_pd])
    cloums = ['Label']
    cloums.extend(continuous_feature)
    cloums.extend(category_feature)
    data.columns = cloums
    return data
```

图 5-11　数据读取函数

如图 5-12 所示 GBDT 预测，首先按照行数拆分了数据集，在训练过程中，首先导入了 lightgbm 包，然后通过使用 LGBMClassifier 初始化了一个树模型，然后使用 fit 函数对树模型进行了训练。

```
def gbdt_predict(data, category_feature):
    for col in category_feature:
        onehot_feats = pd.get_dummies(data[col], prefix = col)
        data.drop([col], axis = 1, inplace = True)
        data = pd.concat([data, onehot_feats], axis = 1)
    # get the first 72 rows == training data
    x_train = data[:72]
    y_train = x_train.pop('Label')
    # get the last 8 rows == testing data
    x_val = data[72:]
    y_val = x_val.pop('Label')

    print('training')
    gbm = lgb.LGBMClassifier(objective='binary',
                             subsample= 0.8,
                             min_child_weight= 0.5,
                             colsample_bytree= 0.7,
                             num_leaves= 20,
                             max_depth = 5,
                             learning_rate=0.01,
                             n_estimators=10000,
                             )
    gbm.fit(x_train, y_train,
            eval_set = [(x_train, y_train), (x_val, y_val)],
            eval_names = ['train', 'val'],
            eval_metric = 'binary_logloss',
            early_stopping_rounds = 10,
            )

    tr_auc = roc_auc_score(y_train, gbm.predict_proba(x_train)[:, 1])
    val_auc = roc_auc_score(y_val, gbm.predict_proba(x_val)[:, 1])
```

图 5-12　GBDT 预测

3. 实验结果

GBDT 取得了 AUC 为 0.69 的模型性能。因为树模型规模的限制，这里仅在 1 万条数据上实现了 GBDT 模型，而 LR 和 FM 模型都在 Criteo 全数据集上完成，因此不适合相互比较。

5.6　梯度提升树＋逻辑回归模型（GBDT＋LR）

5.3 节提到，LR 模型具有计算复杂度低等优点，但是学习能力有限，需要依赖于经验进行特征交叉的设计。而 5.4 节中，FM 模型能够自动学习模型的二阶交叉特征，但是针对二阶以上的特征交叉，FM 模型的计算复杂度太高。5.5 节梳理了 GBDT 的思路，不难发现，GBDT 中，每棵回归树都会将样本集合分成若干类，每次树分裂的过程都利用一种特征对样本集合进行区分。如此，每一个叶子节点对应着高阶特征交叉。这暗含了 CTR 预测对高阶特征交叉的需求。因此，Facebook 于 2014 年创新性地提出了一种新的解决方案，将 GBDT 用来自动生成高阶交叉特征，用来作为 LR 模型的输入特征，也就是本节的 GBDT＋LR[4]。

5.6.1　背景

在计算广告领域，Facebook 每日活跃用户超过 7.5 亿，活跃广告超过一百万。这种数据的规模对 Facebook 来说是一大挑战。囿于特征工程的复杂、FM 不能建模高阶特征交叉的缺点，Facebook 希望使用 GBDT 自动组合特征的性质来完成对高阶特征交叉的挖掘。出于这种考虑，Facebook 提出利用 GBDT 自动学习高阶的特征交叉，代替专家手工设计的交叉特征，并将得到的高阶交叉特征与原始特征组合，用 5.3 节中介绍的 LR 模型训练得到特征到点击率的映射函数，取得了不错的效果，得到了业界的广泛实践。

值得强调的是，GBDT 进行自动的高阶特征交叉和训练 LR 模型是两个各自独立的过程。换言之，GBDT 得到的高阶交叉特征被 LR 模型当作固定输入，不会因为 LR 模型的梯度回传而改变。

5.6.2　模型原理

在特征组合方面，经过研究以后，Facebook 发现 GBDT 模型具有天然的优势。具体地，可以发现多种有区分性的特征以及特征组合，且决策树的路径（叶子节点到根节点的路径）可以被看作高阶特征交叉，省去了人工寻找特征、特征组合的步骤。GBDT＋LR 模型对输入的样本自动进行特征筛选和组合，在树分裂过程时把模型中的每棵树计算得到的预测概率值所属的叶子节点位置记为 1，进而生成新的离散特征向量。

具体而言，对于每个输入样本，根据其特征，必然在每棵二叉树上都会落到一个叶子节点上。每个叶子节点对应着一个高阶的特征交叉。将输入样本对应的叶子节点置为 1，其他叶子节点置为 0，则每个样本在每棵回归树下，都可以得到一个表征高阶特征交叉的独热编码的特征。在图 5-13 中，有两棵回归树，则可以获得两个独热编码（[0,1,0]和[0,1]）代表的特征交叉。接着，将 GBDT 生成的特征和原始的特征拼接，送入

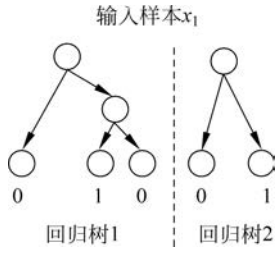

图 5-13　GBDT 自动组合特征

LR 模型中进行权重参数的学习。其中，GBDT 扮演的就是特征提取器的角色，通过 GBDT 自动组合特征的能力解决了 LR 过分依赖人工提取特征的缺陷。

目前为止，有一个问题始终不可以回避："为什么 GBDT 叶子节点的高阶特征交叉可以直接当作 LR 模型的输入？"其实可以用一个比较直观的思路去解释这个问题。GBDT 的目标是学习一个样本的特征到点击率的映射函数，尽量区分开标签不同的样本。每棵回归树得到的高阶特征交叉都根据当前优化目标对样本集合有一定的区分性。

事实上，实践的结果也证明了 GBDT＋LR 比单独的 LR 模型或者 GBDT 模型的性能都要好，大大提高了 CTR 预测的准确率。该方法是早期研究者尝试将不同类型的模型进行融合的典型案例，并为接下来的研究带来了很大的启发，很多研究人员也试图在这种方法的基础上修改模型组件，思考使用更好的分类模型替换 LR，这也就有了后来的 GBDT＋FFM。该模型的第一阶段采取同样的模块，具体地，同样借助 GBDT 对输入的特征自动进行特征筛选和组合。但是第二阶段用 FFM 代替了 LR。用 GBDT 代替手动设计特征交叉，具备了捕获高阶特征组合的能力，同时也推动了特征工程模型化的趋势。

5.3 节～5.6 节介绍了多种用于解决 CTR 预测任务的传统机器学习模型。随着深度学习时代到来，深度学习模型的重要特点——特征表达能力契合了 CTR 预测任务的需求。传统学习模型至少有以下两个缺点。

(1) 模型表达能力不足，不能挖掘出更多潜藏在数据中的模式。

(2) 模型结构固定，不能像深度学习模型一样根据业务场景和数据特点，灵活地调整模型的结构。

5.6.3 实验

1. 数据准备

因为树模型需要全部数据来计算节点分割的位置，而 Criteo 全部的训练集有 4000 万条数据，如果用 GBDT 树模型来对所有数据进行训练，需要的空间开销极大。因此，在 5.5 节(梯度提升树模型)和本节(梯度提升树＋逻辑回归模型)的实验中，我们替代性地使用了 Criteo 部分数据，约 1 万条数据，并按照 8∶2 原则拆分为训练集和测试集，分别用来训练模型和测试模型的性能。

2. 主要代码

我们采取了与 5.5 节相同的数据读取函数，直接从 txt 文件中加载数据进 pandas 的 DataFrame。模型和预测部分如图 5-14 所示。

其中，第一步(GBDT 阶段)，还是初始化树模型，然后训练树模型。不同于 GBDT 预测的是，训练树模型完成后，并没有直接进行测试，而是通过 model.predict 预测模型的叶子节点，也就是树模型带来的特征的高阶组合。

在第二阶段，首先对高阶特征组合和原始特征进行了拼接，然后通过 scikit-learn 模块，初始化一个 LR 模型，然后通过 fit 函数对模型进行了训练。至此，GBDT＋LR 的训练完成，随后进行了测试。

3. 实验结果

GBDT＋LR 在测试集上取得了 AUC 为 0.70 的性能结果。这样的结果对于 GBDT 模型略有提升，但是还有很大的提升空间。在我们的实现中，为了限制内存开销，限定了树模

```
def gbdt_lr_predict(data, category_feature):
    for col in category_feature:
        onehot_feats = pd.get_dummies(data[col], prefix=col)
        data.drop([col], axis=1, inplace=True)
        data = pd.concat([data, onehot_feats], axis=1)
    x_train = data[:72]
    y_train = x_train.pop('Label')
    x_val = data[72:]
    y_val = x_val.pop('Label')
    gbm = lgb.LGBMRegressor(objective='binary',
                            subsample= 0.8,
                            min_child_weight= 0.5,
                            colsample_bytree= 0.7,
                            num_leaves= 32,
                            max_depth = 16,
                            learning_rate=0.01,
                            n_estimators=10)
    gbm.fit(x_train, y_train,
            eval_set = [(x_train, y_train), (x_val, y_val)],
            eval_names = ['train', 'val'],
            eval_metric = 'binary_logloss')
    model = gbm.booster_
    gbdt_feats_train = model.predict(x_train, pred_leaf = True)
    gbdt_feats_test = model.predict(x_val, pred_leaf = True)
    gbdt_feats_name = ['gbdt_leaf_' + str(i) for i in range(gbdt_feats_train.shape[1])]
    df_train_gbdt_feats = pd.DataFrame(gbdt_feats_train, columns = gbdt_feats_name)
    df_test_gbdt_feats = pd.DataFrame(gbdt_feats_test, columns = gbdt_feats_name)
    train = pd.concat([x_train, df_train_gbdt_feats], axis = 1)
    test = pd.concat([x_val, df_test_gbdt_feats], axis = 1)
    train_len = train.shape[0]
    data = pd.concat([train, test])
    gc.collect()
    for col in gbdt_feats_name:
        print('this is feature:', col)
        onehot_feats = pd.get_dummies(data[col], prefix = col)
        data.drop([col], axis = 1, inplace = True)
        data = pd.concat([data, onehot_feats], axis = 1)
    train, test = data[: train_len], data[train_len:]
    gc.collect()
    x_train, x_val = train, test
    lr = LogisticRegression()
    lr.fit(x_train, y_train)
```

图 5-14　GBDT＋LR 预测

型最大的高度为 4，残差树的数目为 30。这样的操作可能限制了模型性能的提升，适当调整这些参数可能进一步提升树模型的挖掘特征高阶交互的性能，进而获得更优的 CTR 预测结果。

5.7　基于深度学习的 CTR 模型

随着深度学习时代的来临，深度学习模型的特征表达能力引起了 CTR 预测的研究人员的密切关注。相比于传统机器学习模型的表达能力不足、模型结构固定等缺点，深度学习模型在图像、语音等领域取得了重要的突破。相关人员尝试结合卷积神经网络（Convolutional Neural Network，CNN）、循环神经网络（Recurrent Neural Network，RNN）等方法进行 CTR 预测。然而，这些方法在点击率预测的问题中并不使用。由于局部感受野的限制，CNN 不适合挖掘特征组合关系；RNN 更适合拟合序列之间的前后依赖关系，而不是全局的组合关系。点击率预测的问题中，用户行为并没有显著的序列关系。

为了将深度学习模型的优势融入 CTR 预测中，相关研究者们进行了融合深度学习技术的探索之旅。研究者们发现将深度的全连接层融合进 CTR 预测的任务中，起到了良好的效果，如 Deep&Cross[10]、PNN[11]、Wide&Deep[5]、DIN[13]、DIEN[12]、DeepFM[6]等，可谓是“一石激起千层浪”。深度学习技术这块“石头”，在点击率预估任务的领域激起了朵朵浪花。在叙述这场探索之旅之前，首先要介绍 CTR 预测模型的记忆能力和泛化能力，分析浅层模型和深度模型的不同，以便针对性地利用不同模型的优势。

5.7.1 模型的记忆能力和泛化能力

首先比较点击率预估领域的传统模型和点击率预估模型的优点。此处采取谷歌应用商店(Google Play)的推荐团队的解释方案，即传统模型拥有更强的“记忆”性，而深度学习结合的模型拥有“泛化”性。

一般而言，传统模型的结构简单，模型倾向于记住历史数据中出现过的共现关系，包含产品或者特征的共现关系。例如，在历史数据中，用户群体经常单击了“泡面”相关的广告后，又单击“火腿肠”相关的广告。在训练过程中，模型记住了“泡面”类的产品和“火腿肠”类的产品经常同时出现。在推荐过程中，用户购买“泡面”之后，模型会继续推荐将“火腿肠”类的产品推荐给用户。“记忆”性就是指分析历史交互数据中，物品之间抑或是特征之间共同出现的关系，并加以利用的能力。

模型的“泛化”性则指的是模型能够对历史数据中不曾出现过的关系进行预测。例如，矩阵分解模型，通过引入隐向量，使数据稀疏的用户或者物品也能生成隐向量，从而获得推荐得分。这是利用全局数据传递到稀疏实例上。深度神经网络因为具有深度发掘数据中潜在的模型的能力，可以多次自动组合数据中的特征，即使推荐阶段遇到的特征组合没有在历史数据中出现过，深度学习模型依然可以给出一个相对较稳定平滑的推荐概率，这就是结合深度学习模型的“泛化”性。

概括而言，基于记忆性的推荐通常更加主题分明，它与用户历史接触过的产品直接相关。另一方面，泛化性指的是根据历史推荐数据中的共现关系，去探索历史数据中很少发生或者没有发生过的特征之间的组合。因为并非直接依托于历史数据，泛化性往往在多样性方面有高的要求。可以看到，在历史数据中出现过的共现关系中，“记忆”性取得了更优秀的效果；对于很少发生或者没有发生过的特征之间的组合，“泛化”性起到的作用高于“记忆”性。如何同时利用“记忆”性和“泛化”性是一个值得关注的问题。

2016 年，谷歌应用商店(Google Play)的推荐团队提供了一种解决方法。该团队提出的 Wide&Deep[5]模型不仅在当时迅速成为业界的主流模型，其最主要的思路——综合 Wide 和 Deep 模型的记忆性和泛化性的想法在今日依然有很大的影响力。

5.7.2 Wide&Deep 模型

Wide&Deep 模型的思路相当简单，既然 Wide 模型拥有更强的“记忆”性，Deep 模型有更强的“泛化”性，那么直接将两种模型融合起来，双路并行就可以同时利用到两种模型的优点。

具体而言，如图 5-15 所示，在 Wide 模型中，研究者们通常对特征执行稀疏的独热(One-hot)编码。除了这样的特征之外，为了实现记忆性，研究者们通常借助特征交叉来对

特征进行处理。例如，用户的手机里安装了微信，通过独热编码来理解“用户手机安装微信”这个特征的值为 1。特征交叉则指代例如 AND(“手机安装微信”，“手机安装 QQ”)，这个交叉特征的值为 1 则代表用户同时满足“手机安装微信”和“手机安装 QQ”。通过这样的手工设计的特征交叉，Wide 模型获得了维度广阔的输入特征，并由此可以记忆用户行为特征中的信息，进而影响 CTR 预估的结果。

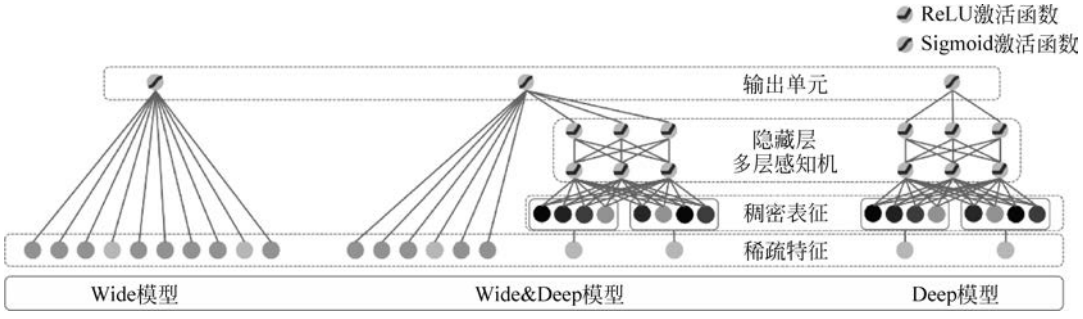

图 5-15　Wide&Deep 的模型框架①

在 Deep 模型部分，研究者们使用了基于嵌入的模型(Embedding-based Model)，这类模型通过学习一个低维度的稠密的向量表征，具体而言，Deep 模型的输入可以是全部的特征向量，经过嵌入(Embedding)层，再对不同的嵌入进行拼接，然后将拼接好的结果送入多层神经网络之中，最终的输出结果送入输出层。Deep 模型具有很强的泛化性，对历史数据没有出现的特征组合同样有效；同时，Deep 模型不需要 Wide 模型的特征工程，成本负担更小。

具体而言，在 Wide 模型的部分，采用了 5.3 节介绍的 LR 模型，该模型的重点在于输入特征的特征工程。通过领域专家的人工设计交叉特征，Wide 模型可以捕捉历史数据中的交叉特征。用 $\phi(x)$表示手工设计的交叉特征，一种常见的特征交叉算子是交叉乘积转换(Cross-product Transformation)，如下。

$$\phi_k(\boldsymbol{x}) = \prod_{i=1}^{d} x_i^{c_{ki}}, \quad c_{ki} \in \{0,1\} \tag{5.9}$$

其中，d 是原始输入的特征的数量，$\boldsymbol{x}=[x_1,x_2,\cdots,x_d]$，$c_{ki}=1$ 则代表第 i 个特征属于第 k 次转换，$\phi(\boldsymbol{x})$则代表手工设计的交叉特征。Wide 模型可以表达为：

$$y^{\text{wide}} = \sigma(\boldsymbol{W}^{\text{wide}}[\boldsymbol{x},\phi(\boldsymbol{x})] + \boldsymbol{b}^{\text{wide}}) \tag{5.10}$$

$\boldsymbol{W}^{\text{wide}}$ 和 $\boldsymbol{b}^{\text{wide}}$ 分别是 Wide 模型的权重矩阵和偏差向量。$\boldsymbol{y}^{\text{wide}}$ 即 Wide 模型预测的 CTR 预测的结果。

同时，Deep 模型可以通过 L 层神经网络将输入的拼接好的嵌入表征映射为预期的输出。其中，第 l 层神经网络的输入到输出如下：

$$z^l = F^l(z^{l-1}) = \sigma(\boldsymbol{W}^l z^{l-1} + \boldsymbol{b}^l) \tag{5.11}$$

其中，F^l 代表第 l 层的映射函数。σ 表示激活函数，z^{l-1} 是第 $l-1$ 层的输出，$\boldsymbol{W}^l$ 是第 l 层神经网络的可训练的权重(Weight)矩阵，维度信息为 $n\times m$，其中，n 表示第 l 层的神经元数量，m 表示第 $l-1$ 层的。而 $\boldsymbol{b}^l$ 表示第 l 层的偏差(bias)向量。若以 x 表示输入的特征，

① https://dl.acm.org/doi/10.1145/2988450.2988454.

E 表示嵌入(Embedding)层,则通过 L 层神经网络和激活函数的最终输出是 $y^{\text{deep}}=z^L=F^L\cdots F^l\cdots F^l(E(x))$。$y^{\text{deep}}$ 即基于深度网络的 CTR 预测结果。

最终模型的输出需要同时考虑浅层模型预测的结果 y^{wide} 和深度模型预测的结果 y^{deep} 的影响,表示为:

$$P(Y=1\mid \boldsymbol{x})=\sigma(\boldsymbol{W}^{\text{wide}}[\boldsymbol{x},\phi(\boldsymbol{x})]+\boldsymbol{W}^{\text{deep}}(E(\boldsymbol{x}))+\boldsymbol{b}) \tag{5.12}$$

其中,$\boldsymbol{W}^{\text{deep}}$ 是一种缩略表达,表达多层神经网络的影响;E 代表嵌入网络,将原始的稀疏特征转换为稠密特征。最后,使用 logistic 损失函数来优化整个模型。

综上所述,Wide&Deep 模型融合了 Wide 模型和 Deep 模型两种模型的优点。如图 5-15 中间部分所示,通过同时利用一个 Wide 模型和一个 Deep 模型,Wide&Deep 模型能够成功应对记忆性和泛化性的挑战:它同时具备两种模型的优势,同时实现记忆性和泛化性。通过在输出单元综合两种模型的决策结果,取得了良好的效果。

如上文所述,Wide&Deep 模型中,Wide 模型采取了 5.3 节提及的 LR 模型,那么,Wide&Deep 模型也继承了其缺点:依赖于耗费成本的人工特征工程。为了解决这个问题,研究者们继续踏上了探索之旅。

5.7.3 DeepFM 模型

Wide&Deep 联合模型同时组合了 Wide 模型记忆性强的优势和 Deep 模型泛化性强的长处,取得了优秀的效果。在 Wide&Deep 模型之中,Wide 模型并没有摆脱特征工程的烦扰:Wide 模型的输入仍需要专家设计的特征交叉信息,这需要消耗大量的时间和精力。

实际中用户单击行为背后的特征之间的各种交互非常复杂。Wide&Deep 使用了联合模型的策略:利用 Wide 模型建模了用户单击行为背后的低阶特征,利用 Deep 模型建模背后的高阶特征。显然存在这样一个结论:同时考虑低阶交叉特征和高阶交叉特征能够提升模型的性能。因此,Huifeng Guo 等人围绕着如何自动构建交叉特征,将 FM 模型和 Wide&Deep 模型的策略综合起来,提出了 DeepFM 模型[6]。这是本书中详细介绍的第二朵深度学习在点击率预估领域激起的“浪花”。

事实上,很多交叉特征可以由领域内的充满经验的专家提出。例子,在炎热的夏天夜晚,吃着烧烤,喝着啤酒,多么惬意!专家观察到这一现象,便会提出“啤酒 & 烧烤”的关联规则。聘请经验更加老到的专家确实是提升点击率预估的一种解决思路,然而,聘请更多的专家也提升了点击率预测任务的成本。

相应地,更多的交叉特征则悄然隐藏在数据之中,很少被人实际中观察到和预知到。如经典的“啤酒 & 尿布”的规则,就是在隐藏的数据中发现了来购买尿布的年轻爸爸们结束了一天的劳累会经常顺手买啤酒这一现象。相比专家通过经验得出的交叉特征,更多的交叉特征是无法被专家预先得知、人工设计的。回顾 5.3 节~5.4 节,不难得出 FM 模型在自动学习二阶特征交叉方面的优秀性能。

针对以上不足,Huifeng Guo 等创新性地将无须执行任何特征工程的建模低阶特征交叉的 FM 结构和建模高阶特征交叉的 DNN 结构融合在了一起,使得提出的新模型避免 Wide&Deep 必需的特征工程。

如图 5-16 所示,与 Wide&Deep 联合模型类似,DeepFM 模型由两个模块组成,FM 模块和 Deep 模块。在输出部分,DeepFM 同样综合了两个模块的输出,表达如下。

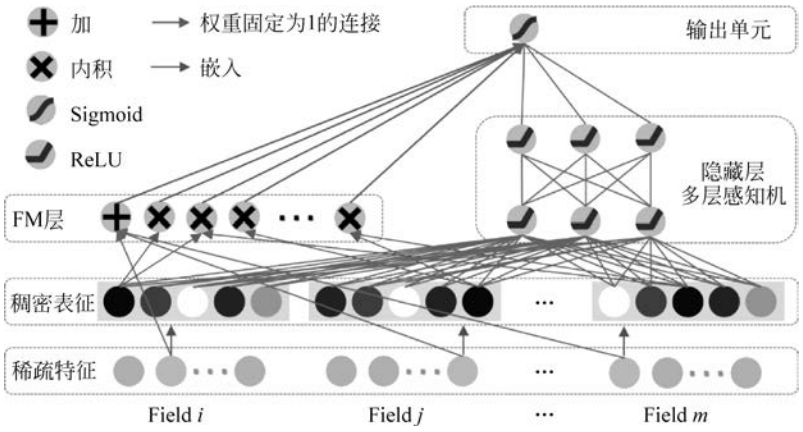

图 5-16　DeepFM 模型结构①

$$\hat{y}=\sigma(y_{\mathrm{FM}}+y_{\mathrm{DNN}}) \tag{5.13}$$

其中，σ 是 Sigmoid 激活函数，将最终的预测值映射到(0,1)的区间。y_{FM} 和 y_{DNN} 分别是 FM 模块和 DNN 模块的输出。

DeepFM 的两个模块之间共享输入。不需要做额外的特征工程。换言之，DeepFM 直接从原始输入数据 $\boldsymbol{x}$ 中同时提取低阶、高阶交叉特征。

具体而言，Deep 模型的部分并没有不同，依然是多层神经网络的整合。但是在低阶交叉特征的获取方面，模型切换成为 FM 模型，该模型具有无须进行特征工程的优势。其计算方式如下。

$$y_{\mathrm{FM}}=<\boldsymbol{w},\boldsymbol{x}>+\sum_{j_1=1}^{d}\sum_{j_2=j_1+1}^{d}<\boldsymbol{V}_i,\boldsymbol{V}_j>x_{j_1}\cdot x_{j_2} \tag{5.14}$$

其中，<>代表内积的计算，$<\boldsymbol{w},\boldsymbol{x}>$ 衡量了特征之间的一阶交互的影响力，对应图中的相加部分；V_i，V_j 分别代表特征 i，j 的隐向量，$<\boldsymbol{V}_i,\boldsymbol{V}_j>$ 表示这两种特征之间的内积，代表这两种特征的交互情况。$\sum_{j_1=1}^{d}\sum_{j_2=j_1+1}^{d}<\boldsymbol{V}_i,\boldsymbol{V}_j>x_{j_1}\cdot x_{j_2}$ 则是在衡量二阶交叉特征的影响力，对应图中的内积部分。可以看到，FM 模型可以自动学习 $\boldsymbol{V}_i$，$\boldsymbol{V}_j$ 参数，可以学习自动的交叉特征；并且，FM 模型的输入只需要 $\boldsymbol{x}$，没有引入任何额外的特征工程。

至此已经讨论了深度学习技术在点击率预估领域激起的两朵浪花：Wide&Deep 模型和 DeepFM 模型。相比于 Wide&Deep 模型，DeepFM 模型已经关注到了可以改进的空间，避免了手工设计交叉特征的高昂成本。

5.7.4　xDeepFM 模型

DeepFM 模型关注到了 Wide 模型部分的不足，然而，对深层网络模块如何利用输入特征，如何学习到高阶特征交叉过程的认识依然不足。对于这个问题，中国科学技术大学、北京邮电大学、微软公司的研究者们进行了探索，联合推出了 xDeepFM 模型[7]。这也是深度学习这块“石头”，在点击率预估领域激起的一朵新的浪花。

① https://dl.acm.org/doi/10.5555/3172077.3172127.

研究者们关注到，在以往运用深层网络模块的过程中，采用拼接嵌入（Embedding）层的特征的方式，一起送入深度网络中。其实这样忽略了不同特征向量的概念，在后续的模块中，对于同一特征向量内的维度依然会计算交互特征信息，即特征自身对应的多个维度也可能被用来计算交叉特征。同时，研究者们还关注到，以往的 DNN 学习的高阶特征交叉是隐式的，因此无法获知特征交叉到什么阶数可以获得最佳的效果；如果能显式地建模高阶交叉特征，能在一定程度上缓解这个问题。

1. xDeepFM 模型原理

如图 5-17 所示，xDeepFM 模型的输入部分是稀疏的特征向量，经过嵌入层，获得特征的稠密表征 $\boldsymbol{X}^{(0)}=[e^1,e^2,\cdots,e^m]^{\mathrm{T}}$。假设每个 field 表征的维度为 d，这样得到的稠密表征就是 $m\times d$ 维的。xDeepFM 模型包含 LR 模型和 DNN 模型的组合。与其他模型不同的是，xDeepFM 模型引入了压缩交互网络（Compressed Interaction Network，CIN）显式地计算高阶交叉特征。该模块可以在向量级别显式地学习高阶的特征交互。具体而言，每增加一层 CIN 网络，特征交叉的阶数就多一层。

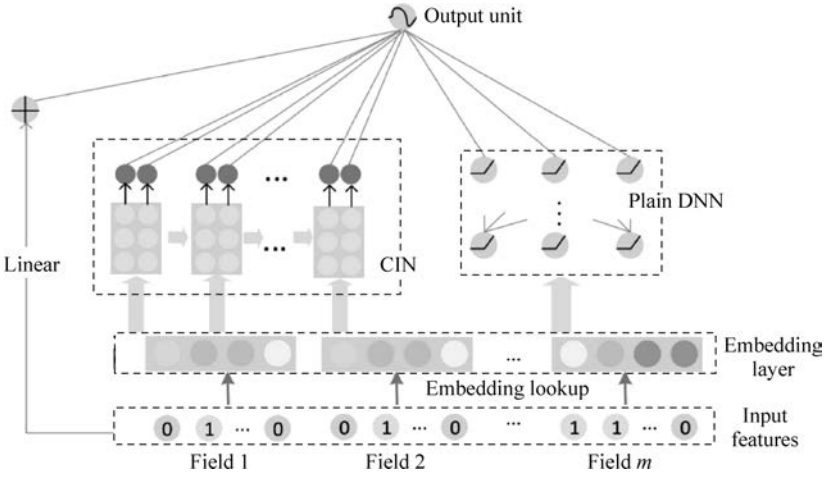

图 5-17　xDeepFM 结构总览①

2. xDeepFM 公式流程

CIN 网络类似于深度网络，可以堆叠深层特征，获得 $X^{(1)}\cdots X^{(k)}$ 的特征，CIN 的第 k 层输出具体表示为：

$$X_{h,*}^{(k)}=\sum_{i=1}^{H_{k-1}}\sum_{j=1}^{m}\boldsymbol{W}_{ij}^{k,h}(X_{i,*}^{(k-1)}\circ X_{j,*}^{(0)}) \tag{5.15}$$

其中，$\boldsymbol{W}^{k,h}$ 是第 h 层特征向量的参数矩阵，$\circ$ 代表哈达玛积（Hadamard Product）。观察式(5.15)不难发现，CIN 的第 k 层输出特征是 $X^{(0)}$ 与 $X^{(k-1)}$ 共同计算的结果。CIN 可以将每一个 field 的特征压缩成一个特征图（Feature Map），这样的一次操作类似于卷积神经网络（Convolutional Neural Network，CNN）中的一个卷积核的作用。通过引入 H_{k+1} 个类似的卷积核，可以将第 k 层的 $H_k\times m$ 个交叉向量压缩到 H_{k+1} 个向量。同时，如图 5-18 所示，CIN 综合每一层的输出并加入池化（Pooling）操作，得到 CIN 输出的交叉特征。

① https://dl.acm.org/doi/abs/10.1145/3219819.3220023.

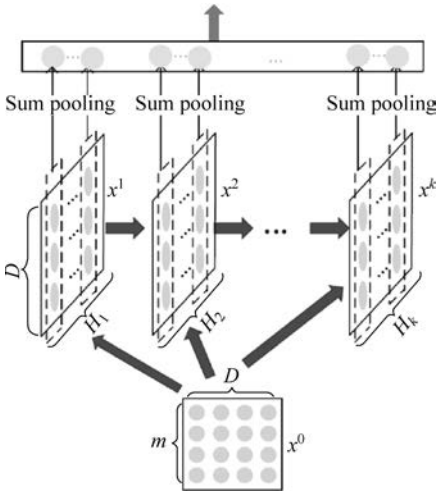

图 5-18　CIN 结构总览[①]

类似 Wide&Deep 联合训练的方式，为了同时利用多种方式得到的特征交叉，在最终的输出阶段，xDeepFM 将线性层交叉的结果（$\boldsymbol{w}_l^{\mathrm{T}}\boldsymbol{x}$）、深度神经网络的结果（$\boldsymbol{x}_d^k$）以及 CIN 网络输出的结果（$\boldsymbol{p}^+$）结合在一起联合训练，如下：

$$\hat{y}=\sigma(\boldsymbol{w}_l^{\mathrm{T}}\boldsymbol{x}+\boldsymbol{w}_{\mathrm{dnn}}^{\mathrm{T}}\boldsymbol{x}_d^k+\boldsymbol{w}_{\mathrm{cin}}^{\mathrm{T}}\boldsymbol{p}^+ + b) \tag{5.16}$$

其中，$\boldsymbol{x}$ 是原始特征，$\boldsymbol{x}_d^k$ 和 $\boldsymbol{p}^+$ 分别是深度神经网络和 CIN 网络的输出结果。整体模型使用 Log-loss 进行优化：

$$\mathcal{L}=-\frac{1}{N}\sum_{i=1}^{N} y_i \log\hat{y}_i + (1-y_i)\log(1-\hat{y}_i) \tag{5.17}$$

虽然 xDeepFM 进一步提升了模型的性能，但是它也存在着明显的缺点。其作者在论文中提出 CIN 的时间复杂度过高，一定程度上会影响推荐系统的实时性。至此，我们已详细介绍了深度学习技术和点击率预估结合的三项经典的工作。在深度学习的潮流下，点击率预估领域激起浪花的代表性工作还有很多，如 FNN、PNN、Deep Crossing 等。然而，深度学习的潮流对于推荐系统的影响远远不止点击率预估这一方面。第 6 章将展示相关领域的研究者们如何将深度学习领域的最前沿技术和推荐系统领域有机结合起来，让推荐系统焕发新的生机。

5.7.5　实验[②]

1. 数据准备

依托于 PaddleRec 的官方代码，我们采取了与 5.3 节相同的数据。

2. 主要代码及实验结果

本节中讨论了 Wide&Deep 模型、DeepFM 模型、xDeepFM 模型。在代码部分（图 5-19～图 5-21），对应地展示三个模型的结构。

相较于简单的 LR 模型，Wide&Deep 模型额外借助了深层神经网络的帮助，具体而言，

① https://dl.acm.org/doi/abs/10.1145/3219819.3220023.

② https://github.com/PaddlePaddle/PaddleRec/tree/release/2.1.0/models/rank.

在代码中体现为 self.__mlp_layers。通过综合不同模块的输出，获得了 AUC 0.82 的结果。这个结果高于 LR 模型的 0.67，这也证明了深度模块的引入对于 CTR 任务是有正面作用的。比较 LR 和 FM 模型，FM 模型获得了 AUC 0.78 的结果，高于 LR 模型，说明 FM 在自动学习二阶交叉特征上的性能给 CTR 预测的性能带来了提升。同时，对比 Wide&Deep 模型（AUC 为 0.82）和 DeepFM 模型（AUC 为 0.78），我们发现 DeepFM 模型反而降低了性能。我们认为原因应该是 DeepFM 更复杂的结构，其复杂的结构容易引发在训练集上的过拟合效应，模型捕捉了一些训练集特有的而测试集并不满足的特征，导致了在测试集上的指标反而下降了。比较 DeepFM 模型和 xDeepFM 模型，xDeepFM 通过替换压缩交互网络模块（CIN），优化了模型结构，提升了 CTR 预测的结果。这充分说明了 CIN 模块能够有效帮助模型提升效果。

表 5-2 总结了本章所有模型在 Criteo 全部数据上取得的结果。在所有模型中，Wide&Deep 模型取得了最优的效果。综合对比传统模型和深度模型，融合了深度神经网络的模型取得了相对更好的结果，深度神经网络对 CTR 预测的提升显露无遗。除表 5-2 外，GBDT 和 GBDT+LR 在 Criteo 的部分数据上分别取得了 AUC 为 0.69 和 0.70 的结果。

表 5-2 CTR 模型性能

Model	LR	FM	Wide&Deep	DeepFM	xDeepFM
AUC	0.67	0.78	0.82	0.78	0.79

```
class WideDeepLayer(nn.Layer):
    def __init__(self, sparse_feature_number, sparse_feature_dim,
                 dense_feature_dim, num_field, layer_sizes):
        super(WideDeepLayer, self).__init__()
        self.sparse_feature_number = sparse_feature_number
        self.sparse_feature_dim = sparse_feature_dim
        self.dense_feature_dim = dense_feature_dim
        self.num_field = num_field
        self.layer_sizes = layer_sizes
        self.wide_part = paddle.nn.Linear(in_features=self.dense_feature_dim,out_features=1,
            weight_attr=paddle.ParamAttr(initializer=paddle.nn.initializer.TruncatedNormal(mean=0.0, std=1.0 / math.sqrt
        use_sparse = True
        self.embedding = paddle.nn.Embedding(
            self.sparse_feature_number,self.sparse_feature_dim,sparse=use_sparse,
            weight_attr=paddle.ParamAttr(name="SparseFeatFactors",initializer=paddle.nn.initializer.Uniform()))
        sizes = [sparse_feature_dim * num_field + dense_feature_dim
                 ] + self.layer_sizes + [1]
        acts = ["relu" for _ in range(len(self.layer_sizes))] + [None]
        self._mlp_layers = []
        for i in range(len(layer_sizes) + 1):
            linear = paddle.nn.Linear(in_features=sizes[i],out_features=sizes[i + 1],
                weight_attr=paddle.ParamAttr(initializer=paddle.nn.initializer.Normal(std=1.0 / math.sqrt(sizes[i]))))
            self.add_sublayer('linear_%d' % i, linear)
            self._mlp_layers.append(linear)
            if acts[i] == 'relu':
                act = paddle.nn.ReLU()
                self.add_sublayer('act_%d' % i, act)
                self._mlp_layers.append(act)
    def forward(self, sparse_inputs, dense_inputs):
        wide_output = self.wide_part(dense_inputs)
        sparse_embs = []
        for s_input in sparse_inputs:
            emb = self.embedding(s_input)
            emb = paddle.reshape(emb, shape=[-1, self.sparse_feature_dim])
            sparse_embs.append(emb)
        deep_output = paddle.concat(x=sparse_embs + [dense_inputs], axis=1)
        for n_layer in self._mlp_layers:
            deep_output = n_layer(deep_output)
        pred = F.sigmoid(paddle.add(x=wide_output, y=deep_output))
        return pred
```

图 5-19 Wide&Deep 模型

```
class DeepFMLayer(nn.Layer):
    def __init__(self, sparse_feature_number, sparse_feature_dim,
                 dense_feature_dim, sparse_num_field, layer_sizes):
        super(DeepFMLayer, self).__init__()
        self.sparse_feature_number = sparse_feature_number
        self.sparse_feature_dim = sparse_feature_dim
        self.dense_feature_dim = dense_feature_dim
        self.sparse_num_field = sparse_num_field
        self.layer_sizes = layer_sizes
        self.fm = FM(sparse_feature_number, sparse_feature_dim,
                     dense_feature_dim, sparse_num_field)
        self.dnn = DNN(sparse_feature_number, sparse_feature_dim,
                       dense_feature_dim, dense_feature_dim + sparse_num_field,
                       layer_sizes)
        self.bias = paddle.create_parameter(shape=[1],dtype='float32',
            default_initializer=paddle.nn.initializer.Constant(value=0.0))
    def forward(self, sparse_inputs, dense_inputs):
        y_first_order, y_second_order, feat_embeddings = self.fm(sparse_inputs,dense_inputs)
        y_dnn = self.dnn(feat_embeddings)
        predict = F.sigmoid(y_first_order + y_second_order + y_dnn)
        return predict
```

图 5-20　DeepFM 模型

```
class xDeepFMLayer(nn.Layer):
    def __init__(self, sparse_feature_number, sparse_feature_dim,
                 dense_feature_dim, sparse_num_field, layer_sizes_cin,layer_sizes_dnn):
        super(xDeepFMLayer, self).__init__()
        self.sparse_feature_number = sparse_feature_number
        self.sparse_feature_dim = sparse_feature_dim
        self.dense_feature_dim = dense_feature_dim
        self.sparse_num_field = sparse_num_field
        self.layer_sizes_cin = layer_sizes_cin
        self.layer_sizes_dnn = layer_sizes_dnn
        self.fm = Linear(sparse_feature_number, sparse_feature_dim,dense_feature_dim, sparse_num_field)
        self.cin = CIN(sparse_feature_dim,dense_feature_dim + sparse_num_field, layer_sizes_cin)
        self.dnn = DNN(sparse_feature_dim,dense_feature_dim + sparse_num_field, layer_sizes_dnn)
        self.bias = paddle.create_parameter(shape=[1],dtype='float32',
            default_initializer=paddle.nn.initializer.Constant(value=0.0))
    def forward(self, sparse_inputs, dense_inputs):
        y_linear, feat_embeddings = self.fm(sparse_inputs, dense_inputs)
        y_cin = self.cin(feat_embeddings)
        y_dnn = self.dnn(feat_embeddings)
        predict = F.sigmoid(y_linear + self.bias + y_cin + y_dnn)
        return predict
```

图 5-21　xDeepFM 模型

5.8　本章小结

本章从推荐系统的“召回层”和“排序层”开始，以排序层的重要任务——点击率预测为中心，介绍了点击率预测任务的发展脉络。5.2 节简单地给出了本章介绍的模型之间联系的框图。5.3 节～5.7 节详细介绍了点击率预测的一些经典模型，以及结合了深度学习优势的组合模型。在 5.8 节，如表 5-3 所示，总结了对 5.3 节～5.7 节的所有模型，梳理不同 CTR 模型的优缺点，帮助读者更深刻地理解 CTR 预测。

表 5-3　CTR 模型小结

模　型	简　述	优　点	缺　点
LR	给每个特征赋上可学习的权重	简单、可解释性强	需要特征工程，成本高
FM	对二阶权重矩阵进行分解	自动学习二阶特征交叉	高阶交叉情况，复杂度过高
GBDT	级联多个回归树构成强学习器	高阶特征交叉	无法给每个特征赋予权重

续表

模　型	简　述	优　点	缺　点
GBDT＋LR	用 GBDT 生成新的离散特征	利用 GBDT 生成高阶特征交叉	模型表达能力不足
Wide&Deep	同时利用 LR 和 DNN	综合 LR 和 DNN 的优势	需要特征工程，成本高
DeepFM	同时利用 FM 和 DNN	综合 FM 和 DNN 的优势	DNN 部分隐式建模，解释性差
xDeepEM	显式地学习高阶特征交叉	可解释性强，易于调整	CIN 模块的时间复杂度较高

参考文献

[1] McMahan H B, Holt G, Sculley D, et al. Ad click prediction: a view from the trenches[C]//Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2013: 1222-1230.

[2] Rendle S. Factorization machines[C]//IEEE International Conference on data mining. IEEE, 2010: 995-1000.

[3] Ke G, Meng Q, Finley T, et al. Lightgbm: A highly efficient gradient boosting decision tree[J]. Advances in neural information processing systems, 2017, 30: 3146-3154.

[4] He X, Pan J, Jin O, et al. Practical lessons from predicting clicks on ads at facebook[C]//Proceedings of the Eighth International Workshop on Data Mining for Online Advertising. 2014: 1-9.

[5] Cheng H T, Koc L, Harmsen J, et al. Wide & deep learning for recommender systems[C]//Proceedings of the 1st workshop on deep learning for recommender systems. 2016: 7-10.

[6] Guo H, Tang R, Ye Y, et al. DeepFM: a factorization-machine based neural network for CTR prediction[C]//Proceedings of the 26th International Joint Conference on Artificial Intelligence. 2017: 1725-1731.

[7] Lian J, Zhou X, Zhang F, et al. xdeepfm: Combining explicit and implicit feature interactions for recommender systems[C]//Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2018: 1754-1763.

[8] Trofimov I, Kornetova A, Topinskiy V. Using boosted trees for click-through rate prediction for sponsored search[C]//Proceedings of the Sixth International Workshop on Data Mining for Online Advertising and Internet Economy. 2012: 1-6.

[9] Juan Y, Zhuang Y, Chin W S, et al. Field-aware factorization machines for CTR prediction[C]//Proceedings of the 10th ACM Conference on recommender systems. 2016: 43-50.

[10] Wang R, Fu B, Fu G, et al. Deep & cross network for ad click predictions[M]//Proceedings of the ADKDD'17. 2017: 1-7.

[11] Qu Y, Cai H, Ren K, et al. Product-based neural networks for user response prediction[C]//2016 IEEE 16th International Conference on Data Mining (ICDM). IEEE, 2016: 1149-1154.

[12] Zhou G, Mou N, Fan Y, et al. Deep interest evolution network for click-through rate prediction[C]//Proceedings of the AAAI Conference on artificial intelligence. 2019, 33(01): 5941-5948.

[13] Zhou G, Zhu X, Song C, et al. Deep interest network for click-through rate prediction[C]//Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2018: 1059-1068.

[14] Su X, Khoshgoftaar T M. A survey of collaborative filtering techniques[J]. Advances in artificial intelligence, 2009.

[15] Chang Y W, Hsieh C J, Chang K W, et al. Training and testing low-degree polynomial data mappings via linear SVM[J]. Journal of Machine Learning Research, 2010, 11(4).