DeepFM理论与其应用

原创 vent 机器史学 2019-05-25

DeepFM[1]是哈工大Guo博士在华为诺亚实验室实习期间，提出的一种深度学习方法，它基于Google的经典论文Wide&Deep learning 基础上，通过将原论文的wide部分--LR部分替换成FM[4]，从而改进了原模型依然需要人工特征工程的缺点，得到一个end to end 的深度学习模型。DeepFM在企业数据集(华为应用商店)和公开数据集(criteo)上都取得不错的效果，目前该方法在不少互联网公司的推荐、广告系统中得到了较为广泛的应用。

1. CTR预估中的特征分析

在CTR预测中，挖掘用户行为中的隐藏特征以及它们之间的交叉特征已经成为推荐算法中最核心的一部分。华为通过对自己应用市场的用户进行行为分析，得到以下两个重要的结论。

1. 用户喜欢在等待外卖送达的时段下载APP。它说明时间和APP类别的二维特征交叉是一个有效的特征输入信号。
2. 青少年的男性用户喜欢下载射击或者RPG游戏APP。它则说明年龄、性别、APP类别的三维特征交叉也是一个有效的输入信号。

我们可以看到，CTR预估中主要挑战是有效对特征交互建模，有些特征交互可以很容易理解，因此特征工程的专家可以人工设计出来。然而，绝大部分特征都是隐藏在数据背后，难以形成专家的先验知识，只能通过机器学习自动生成。由于实际应用中使用到的特征非常多(原始特征经常有几十到上百维)，就算是简单的特征交互，专家其实也无法对全部特征交叉进行有效建模。

广义线性模型实现简单、性能好，但是缺乏学习特征交叉的能力，通常在工业实践中会人工做特征工程来解决这个问题。FM采用隐向量(latent vector)的内积作为对特征交叉的建模方法，具有很好的效果，FM在深度学习时代之前是CTR预估最为广泛应用的一种算法，它在实践中通常只会利用二维的特征交叉。

总而言之，用户行为的特征维度是高度复杂的，无论是低维还是高维的特征交叉都会起到重要的作用。根据Google wide&deep model，它在建模过程中同时考虑了低维和高维的特征交叉，以提升模型的效果。

2.深度学习在CTR预估的进展

Google的Wide & Deep Learning for Recommender Systems[2]是深度学习应用于推荐、广告等CTR领域的重要论文。过去几年，神经网络已经在图像、音频等领域得到广泛应用，而由于推荐、广告等领域由于数据的稀疏性、离散性，无法直接套用传统的深度学习模型。

基于深度学习的思想，Google 提出一种深度模块和广度模块结合的神经网络模型。Wide端使用常见的LR(FTRL[5]实现)模型，将常见的离散特征、低维特征组合作为输入，实现了模型的记忆能力。换句话说，模型能够很好记住用户的喜好，给用户推荐 **常见喜好**的内容。Deep端将离散特征通过embedding方法转化成稠密特征向量输入，实际上实现了tag向量的模糊查询，扩充了模型的泛化能力。换句话说，模型能够更好理解用户-物品之间内在的高维关系，给用户推荐 **罕见但是可能喜好** 的内容，破解“信息茧房”的问题。

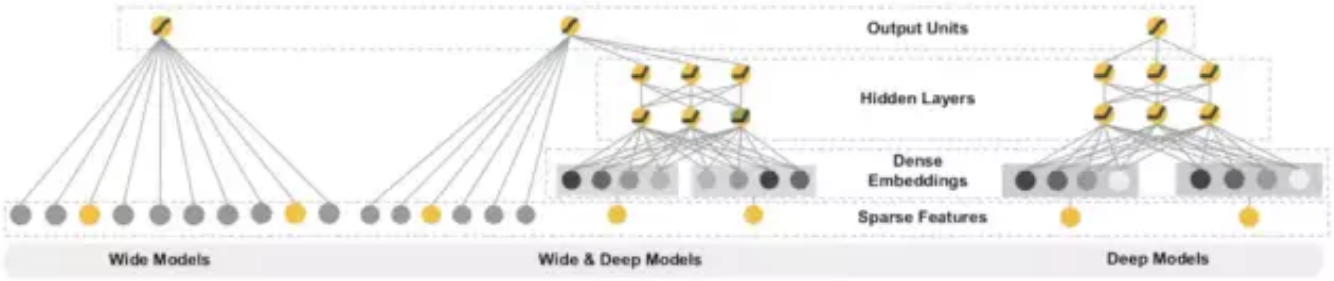


Figure 1: The spectrum of Wide & Deep models.

Wide&Deep

2.1 稀疏特征的优点：

- LR, DNN在底层还是一个线性模型，但是现实生活中，标签y与特征x之间较少存在线性关系，而往往是分段的。以"点击率 ~ 历史曝光次数" 之间的关系为例，之前曝光过1、2次的时候，“点击率 ~ 历史曝光次数”之间一般是正相关的，再多曝光1、2次，用户由于好奇，没准就点击了；但是，如果已经曝光过8、9次了，由于用户已经失去了新鲜感，越多曝光，用户越不可能再点，这时“点击率 ~ 历史曝光次数”就表现出负相关性。因此，categorical特征相比于numeric特征，更加符合现实场景。
- 推荐、搜索一般都是基于用户、商品的标签画像系统，而标签天生就是categorical的
- 稀疏的类别/ID类特征，可以稀疏地存储、传输、运算，提升运算效率。

2.2 稀疏特征的缺点：

- 稀疏的categorical/ID类特征，也有着单个特征表达能力弱、特征组合爆炸、分布不均匀导致受训程度不均匀的缺点。
- FTRL 充分输入的稀疏性在线更新模型，训练出的模型也是稀疏的，便于快速预测。

- Parameter Server，充分利用特征的稀疏性，不必在各机器之间同步全部模型，而让每台机器“按需”同步自己所需要的部分模型权重，“按需”上传这一部分权重的梯度。
- TensorFlow Feature Column类，除了一个numeric_column是处理实数特征的，其实的都是围绕处理categorical特征的，封装了常见的分桶、交叉、哈希等操作。

总而言之：

- Wide for Memorization，wide侧记住的是历史数据中那些常见、高频的模式。根据人工经验、业务背景，将有价值的、显而易见的特征及特征组合输入wide侧。

- Deep for Generation，deep侧通过embedding将tag向量化，变tag的精确匹配，为tag向量的模糊查询，因而模型具备良好的“扩展”能力。

Wide & Deep模型应用Google Play的数据，它包含超过10亿活跃用户以及上百万的app行为。在线实验显示Wide& Deep model 有效提升了App的购买率。代码开源集成到了TensorFlow内，调用DNNLinearCombinedClassifier 这个estimator就可以。

```
estimator = DNNLinearCombinedClassifier(
    # wide侧设置
    linear_feature_columns=[categorical_feature_a_x_categorical_feature_b],
    linear_optimizer=tf.train.FtrlOptimizer(...),
    # deep侧设置
    dnn_feature_columns=[
        categorical_feature_a_emb, categorical_feature_b_emb,
        numeric_feature],
    dnn_hidden_units=[1000, 500, 100],
    dnn_optimizer=tf.train.ProximalAdagradOptimizer(...),
    # warm-start 设置
    warm_start_from="/path/to/checkpoint/dir")
```

除了Wide and Deep 以外还有数篇文章探索深度学习在CTR预估领域的应用，其中包括采用FM对特征做初始化处理的FNN[3]。FNN通过
它的模型如下图所示：

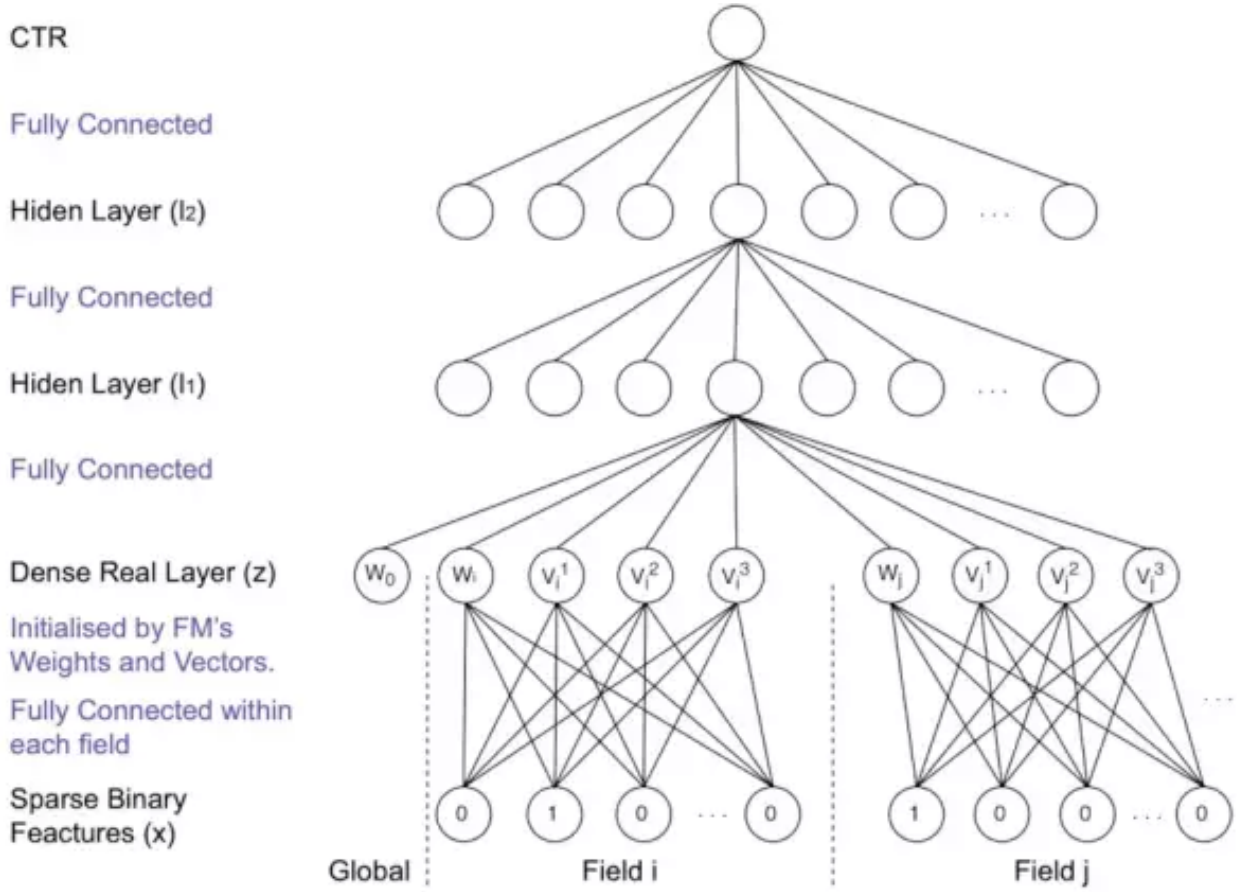
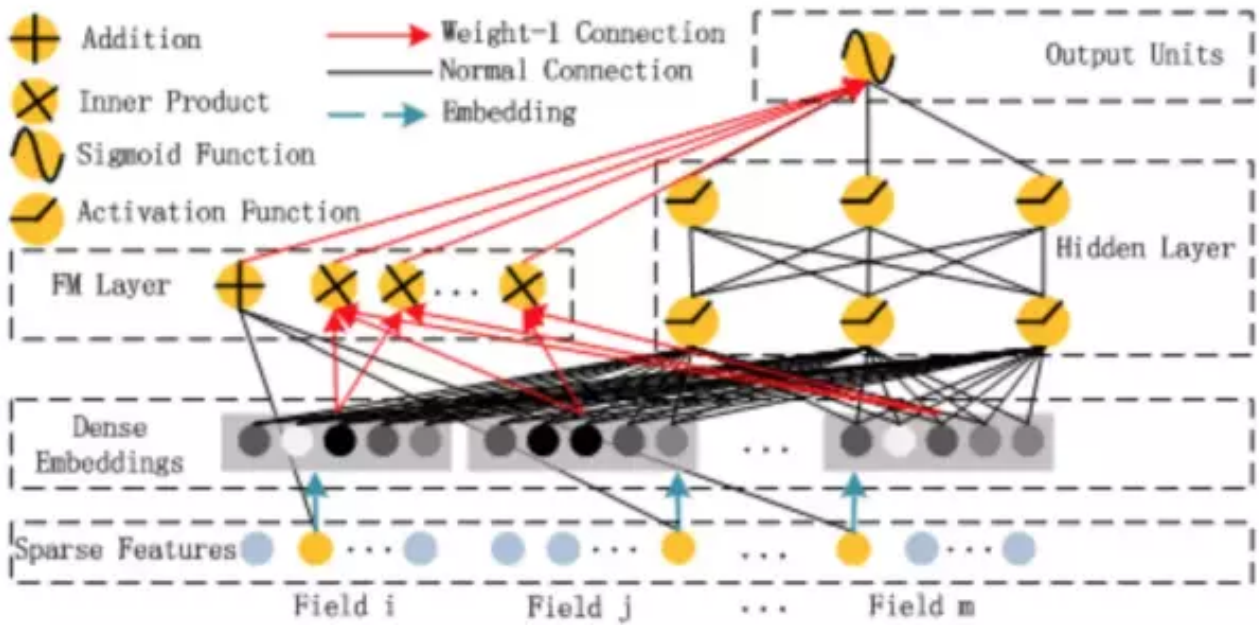


Fig. 1. A 4-layer FNN model structure.

FNN
它主要缺点在于，embedding 后的特征可能会被FM模型过度影响。使用FM对特征做预处理的做法，可能影响了模型的性能和效率。它只能刻画高维的特征交互，而不像Wide & Deep那样高维和低维特征交叉都能刻画到。

3.DeepFM核心思想

DeepFM将Wide and Deep 模型中的Wide侧的LR替换成FM，克服了原有模型依然需要对低维特征做特征工程的缺点，实现了一个无需任何人工特征工程的end to end 模型。DeepFM在wide侧和deep侧共享了embedding的特征向量。



DeepFM架构

可以看到DeepFM的数学形式化：

$$y = sigmod(yFM + yDNN)$$

yFM 是FM组件的输出，yDNN是深度组件的输出结果。FM组件能够捕获一维特征的同时，还能很好捕获二维稀疏组合特征。如下图所示：

$$\hat{y}(\mathbf{x}) := w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j$$

FM数学公式

yDNN旨在学习高维特征组合，和图像、音频的稠密数值张量不同的是，在推荐系统中DNN模型的数据输入通常都是非常稀疏的张量，所以在技术上一般会采用embedding层来压缩数据空间维度。

DeepFM 在企业数据集(华为应用商店)和公开数据集(criteo)进行多次实验，采用AUC和LogLoss来评估效果。具体效果如下图所示：

Table 2: Performance on CTR prediction.

	Company*		Criteo	
	AUC	LogLoss	AUC	LogLoss
LR	0.8641	0.02648	0.7804	0.46782
FM	0.8679	0.02632	0.7894	0.46059
FNN	0.8684	0.02628	0.7959	0.46350
IPNN	0.8662	0.02639	0.7971	0.45347
OPNN	0.8657	0.02640	0.7981	0.45293
PNN*	0.8663	0.02638	0.7983	0.45330
LR & DNN	0.8671	0.02635	0.7858	0.46596
FM & DNN	0.8658	0.02639	0.7980	0.45343
DeepFM	**0.8715**	**0.02619**	**0.8016**	**0.44985**

评估结果

DeepFM在公开数据上，比LR&DNN AUC提升了一百多个基点，是一个非常好的改进。

4. DeepFM重要参数

这篇文章有趣的部分是探索与分享整个模型的多个超参，从而分析如何得到一个更好效果的模型。

4.1 激活函数

relu 函数和 tanh 函数比sigmod函数效果更好。

4.2 Dropout

下图效果显示：采用适合的随机性能够加强模型的鲁棒性，建议采用dropout比率在0.6~0.9之间。

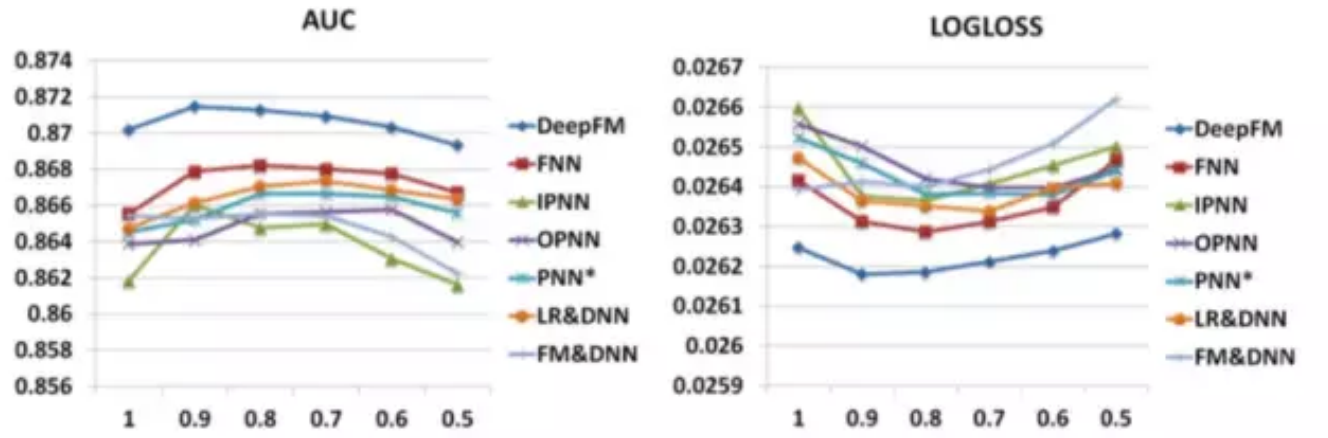


Figure 8: AUC and Logloss comparison of dropout.

Dropout

4.3 每层神经元个数

建议采用200~400个神经元能够给模型更好效果。

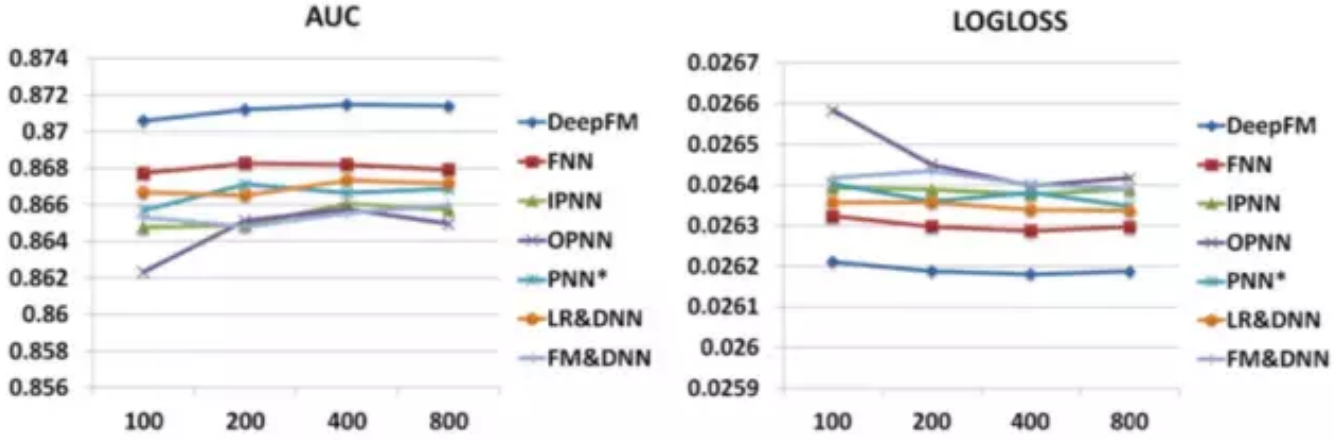


Figure 9: AUC and Logloss comparison of number of neurons.

神经元个数

4.4 隐含层数量

增加隐含层的数量能够一定程度提升模型效果，但是要注意过拟合的情况。建议3~5个隐藏层为妙。

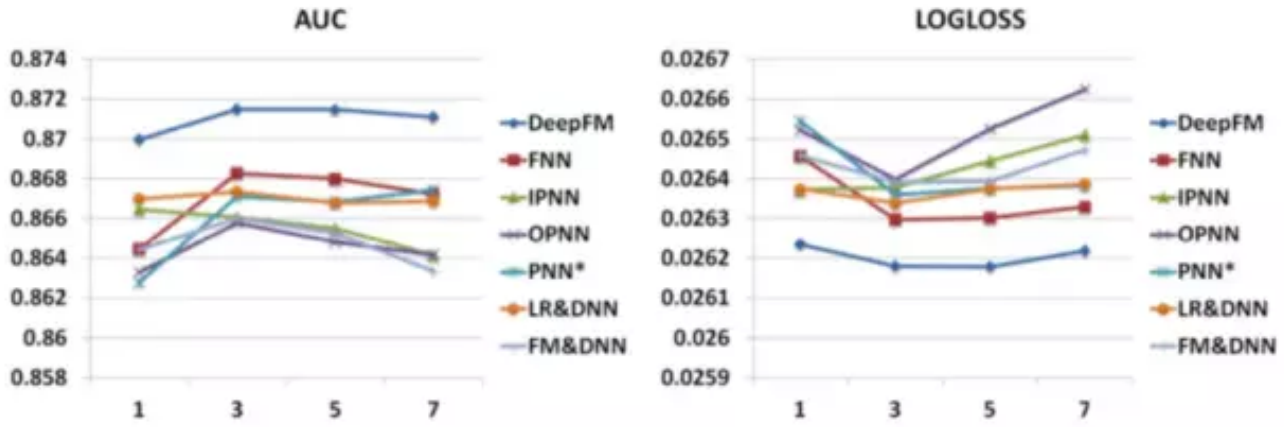


Figure 10: AUC and Logloss comparison of number of layers.

隐藏层数量

4.5 网络结构

文章中测试了四种深度网络结构，不变型(constant),增长型(increasing),衰减型(decreasing),钻石型(diamond)。文章保证四种网络结构神经元总量一致，采用三层隐藏层，从而四种形状具体为：constant (200-200-200), increasing (100- 200-300), decreasing (300-200-100), and diamond (150-300- 150).
如下图所示，constant型效果更好。这点比较有意思，因为在Wide & Deep Model中，采用的是decreasing型。网络结构的效果也取决于实验数据本身。

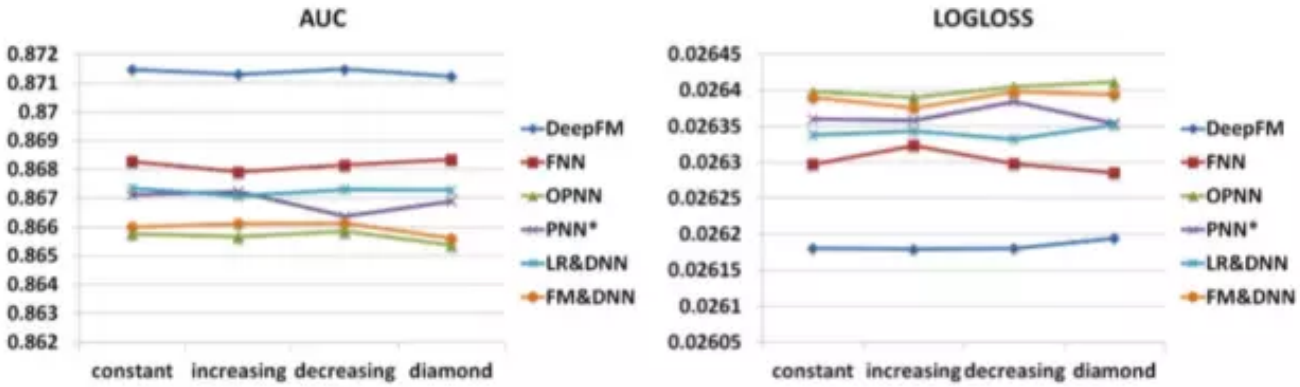


Figure 11: AUC and Logloss comparison of network shape.

网络结构

5. DeepFM的实现

DeepCTR[6]是一个实现了多种深度CTR预估模型的python库，下面引用它基于criteo数据，所实现的DeepFM样例代码

```
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, MinMaxScaler

from deepctr.models import DeepFM
from deepctr.utils import SingleFeat

if __name__ == "__main__":
    data = pd.read_csv('./criteo_sample.txt')

    #拆分稀疏和稠密特征
    sparse_features = ['C' + str(i) for i in range(1, 27)]
    dense_features = ['I' + str(i) for i in range(1, 14)]

    data[sparse_features] = data[sparse_features].fillna('-1', )
    data[dense_features] = data[dense_features].fillna(0, )
    target = ['label']

    # 1.类别特征的编码与稠密特征做归一化
    for feat in sparse_features:
        lbe = LabelEncoder()
        data[feat] = lbe.fit_transform(data[feat])
    mms = MinMaxScaler(feature_range=(0, 1))
    data[dense_features] = mms.fit_transform(data[dense_features])

    # 2.统计稀疏特征类别特征个数，记录稠密特征类目
    sparse_feature_list = [SingleFeat(feat, data[feat].nunique())
                           for feat in sparse_features]
    dense_feature_list = [SingleFeat(feat, 0,)
                          for feat in dense_features]

    # 3.生成模型输入特征

    train, test = train_test_split(data, test_size=0.2)
    train_model_input = [train[feat.name].values for feat in sparse_feature_list] + \
                        [train[feat.name].values for feat in dense_feature_list]
    test_model_input = [test[feat.name].values for feat in sparse_feature_list] + \
                       [test[feat.name].values for feat in dense_feature_list]

    # 4.定义模型、预测、评估模型
    model = DeepFM({"sparse": sparse_feature_list,
                    "dense": dense_feature_list}, task='binary')
    model.compile("adam", "binary_crossentropy",
                  metrics=['binary_crossentropy'], )

    history = model.fit(train_model_input, train[target].values,
                        batch_size=256, epochs=10, verbose=2, validation_split=0.2, )
    pred_ans = model.predict(test_model_input, batch_size=256)
    print("test LogLoss", round(log_loss(test[target].values, pred_ans), 4))
    print("test AUC", round(roc_auc_score(test[target].values, pred_ans), 4))
```

引用