推荐系统遇上深度学习(十)--GBDT+LR融合方案实战

原创 文文 小小挖掘机 2018-05-19

来自专辑

推荐系统遇上深度学习

写在前面的话

GBDT和LR的融合在广告点击率预估中算是发展比较早的算法，为什么会在这里写这么一篇呢？本来想尝试写一下阿里的深度兴趣网络(Deep Interest Network)，发现阿里之前还有一个算法MLR，然后去查找相关的资料，里面提及了树模型也就是GBDT+LR方案的缺点，恰好之前也不太清楚GBDT+LR到底是怎么做的，所以今天我们先来了解一下GBDT和LR的融合方案。

1、背景

在CTR预估问题的发展初期，使用最多的方法就是逻辑回归(LR)，LR使用了Sigmoid变换将函数值映射到0~1区间，映射后的函数值就是CTR的预估值。

LR属于线性模型，容易并行化，可以轻松处理上亿条数据，但是学习能力十分有限，需要大量的特征工程来增加模型的学习能力。但大量的特征工程耗时耗力同时并不一定会带来效果提升。因此，如何自动发现有效的特征、特征组合，弥补人工经验不足，缩短LR特征实验周期，是亟需解决的问题。

FM模型通过隐变量的方式，发现两两特征之间的组合关系，但这种特征组合仅限于两两特征之间，后来发展出来了使用深度神经网络去挖掘更高层次的特征组合关系。但其实在使用神经网络之前，GBDT也是一种经常用来发现特征组合的有效思路。

Facebook 2014年的文章介绍了通过GBDT解决LR的特征组合问题，随后Kaggle竞赛也有实践此思路，GBDT与LR融合开始引起了业界关注。

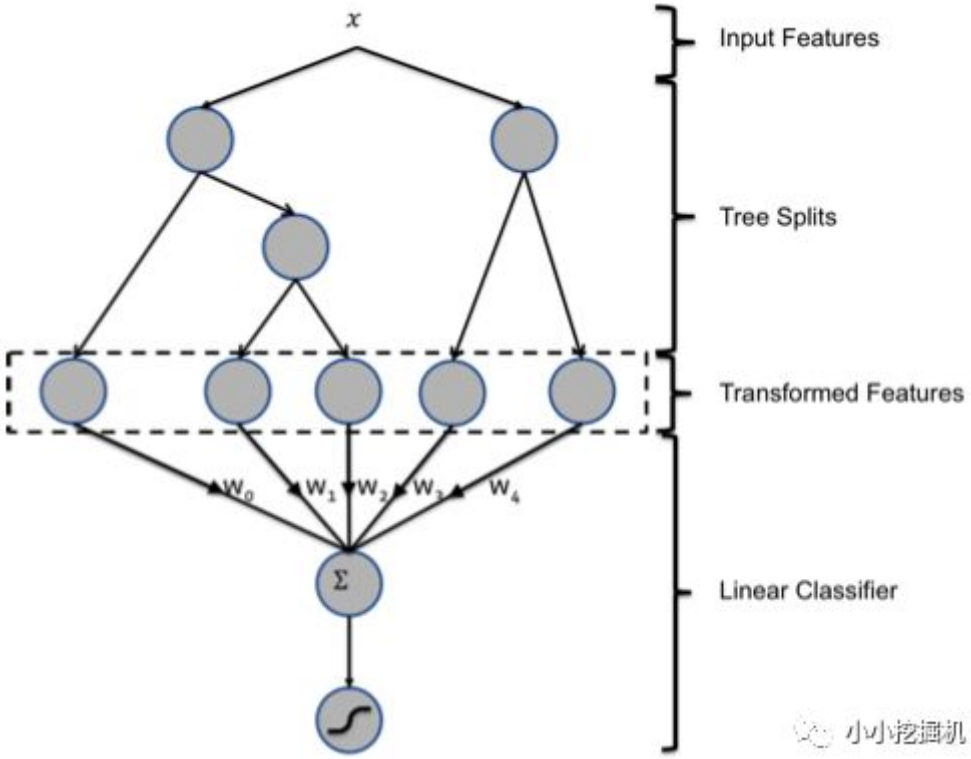
在介绍这个模型之前，我们先来介绍两个问题：
1）为什么要使用集成的决策树模型，而不是单棵的决策树模型：一棵树的表达能力很弱，不足以表达多个有区分性的特征组合，多棵树的表达能力更强一些。可以更好的发现有效的特征和特征组合
2）为什么建树采用GBDT而非RF：RF也是多棵树，但从效果上有实践证明不如GBDT。且GBDT前面的树，特征分裂主要体现对多数样本有区分度的特征；后面的树，主要体现的是经

过前N颗树，残差仍然较大的少数样本。优先选用在整体上有区分度的特征，再选用针对少数样本有区分度的特征，思路更加合理，这应该也是用GBDT的原因。

了解了为什么要用GBDT，我们就来看看到底二者是怎么融合的吧！

2、GBDT和LR的融合方案

GBDT和LR的融合方案，FaceBook的paper中有个例子：



图中共有两棵树，x为一条输入样本，遍历两棵树后，x样本分别落到两颗树的叶子节点上，每个叶子节点对应LR一维特征，那么通过遍历树，就得到了该样本对应的所有LR特征。构造的新特征向量是取值0/1的。举例来说：上图有两棵树，左树有三个叶子节点，右树有两个叶子节点，最终的特征即为五维的向量。对于输入x，假设他落在左树第一个节点，编码[1,0,0]，落在右树第二个节点则编码[0,1]，所以整体的编码为[1,0,0,0,1]，这类编码作为特征，输入到LR中进行分类。

这个方案还是很简单的吧，在继续介绍下去之前，我们先介绍一下代码实践部分。

3、GBDT+LR代码实践

本文介绍的代码只是一个简单的Demo，实际中大家需要根据自己的需要进行参照或者修改。

github 地 址 : https://github.com/princewen/tensorflow_practice/tree/master/recommendation/GBDT%2BLR-Demo

训练GBDT模型

本文使用lightgbm包来训练我们的GBDT模型，训练共100棵树，每棵树有64个叶子结点。

```
df_train = pd.read_csv('data/train.csv')
df_test = pd.read_csv('data/test.csv')

NUMERIC_COLS = [
    "ps_reg_01", "ps_reg_02", "ps_reg_03",
    "ps_car_12", "ps_car_13", "ps_car_14", "ps_car_15",
]

print(df_test.head(10))

y_train = df_train['target']  # training label
y_test = df_test['target']  # testing label
X_train = df_train[NUMERIC_COLS]  # training dataset
X_test = df_test[NUMERIC_COLS]  # testing dataset

# create dataset for lightgbm
lgb_train = lgb.Dataset(X_train, y_train)
lgb_eval = lgb.Dataset(X_test, y_test, reference=lgb_train)

params = {
    'task': 'train',
    'boosting_type': 'gbdt',
    'objective': 'binary',
    'metric': {'binary_logloss'},
    'num_leaves': 64,
    'num_trees': 100,
    'learning_rate': 0.01,
    'feature_fraction': 0.9,
    'bagging_fraction': 0.8,
    'bagging_freq': 5,
    'verbose': 0
}

# number of leaves,will be used in feature transformation
num_leaf = 64

print('Start training...')
# train
gbm = lgb.train(params,
                lgb_train,
                num_boost_round=100,
                valid_sets=lgb_train)

print('Save model...')
# save model to file
gbm.save_model('model.txt')

print('Start predicting...')
# predict and get data on leaves, training data
```

特征转换

在训练得到100棵树之后，我们需要得到的不是GBDT的预测结果，而是每一条训练数据落在了每棵树的哪个叶子结点上，因此需要使用下面的语句：

```
y_pred = gbm.predict(X_train, pred_leaf=True)
```

打印上面结果的输出，可以看到shape是(8001,100)，即训练数据量*树的棵树

```
print(np.array(y_pred).shape)
print(y_pred[0])
```

结果为：

```
(8001, 100)
[[43 26 47 47 47 19 36 19 50 52 29  0  0  0 46 23 13 27 27 13 10 22  0 10
   4 57 17 55 54 57 59 42 22 22 22 13  8  5 27  5 58 23 58 14 16 16 10 32
  60 32  4  4  4  4  4 46 57 48 57 34 54  6 35  6  4 55 13 23 15 51 40  0
  47 40 10 29 24 24 31 24 55  3 41  3 22 57  6  0  6  6 57 55 57 16 12 18
  30 15 17 30]]
```

然后我们需要将每棵树的特征进行one-hot处理，如前面所说，假设第一棵树落在43号叶子结点上，那我们需要建立一个64维的向量，除43维之外全部都是0。因此用于LR训练的特征维数共num_trees * num_leaves。

```
print('Writing transformed training data')
transformed_training_matrix = np.zeros([len(y_pred), len(y_pred[0]) * num_leaf],
                                       dtype=np.int64)  # N * num_tress * num_leafs
for i in range(0, len(y_pred)):
    temp = np.arange(len(y_pred[0])) * num_leaf + np.array(y_pred[I])
    transformed_training_matrix[i][temp] += 1
```

当然，对于测试集也要进行同样的处理:

```
y_pred = gbm.predict(X_test, pred_leaf=True)
print('Writing transformed testing data')
transformed_testing_matrix = np.zeros([len(y_pred), len(y_pred[0]) * num_leaf], dtype
=np.int64)
for i in range(0, len(y_pred)):
    temp = np.arange(len(y_pred[0])) * num_leaf + np.array(y_pred[I])
    transformed_testing_matrix[i][temp] += 1
```

LR训练

然后我们可以用转换后的训练集特征和label训练我们的LR模型，并对测试集进行测试：

```
lm = LogisticRegression(penalty='l2',C=0.05) # logestic model construction
lm.fit(transformed_training_matrix,y_train)  # fitting the data
y_pred_test = lm.predict_proba(transformed_testing_matrix)   # Give the probabilty on
each label
```

我们这里得到的不是简单的类别，而是每个类别的概率。

效果评价

在Facebook的paper中，模型使用NE(Normalized Cross-Entropy)，进行评价，计算公式如下:

$$NE = \frac{-\frac{1}{N}\sum_{i=1}^{n}(\frac{1+y_i}{2}log(p_i) + \frac{1-y_i}{2}log(1-p_i))}{-(p*log(p)+(1-p)*log(1-p))}$$

代码如下:

```
NE = (-1) / len(y_pred_test) * sum(((1+y_test)/2 * np.log(y_pred_test[:,1]) +  (1-y_t
est)/2 * np.log(1 - y_pred_test[:,1])))
print("Normalized Cross Entropy " + str(NE))
```

4、反思

现在的GBDT和LR的融合方案真的适合现在的大多数业务数据么？现在的业务数据是什么？是大量离散特征导致的高维度离散数据。而树模型对这样的离散特征，是不能很好处理的，要说为什么，因为这容易导致过拟合。下面的一段话来自知乎:

> 假设有1w 个样本， y类别0和1，100维特征，其中10个样本都是类别1，而特征 f1的值为0，1，且刚好这10个样本的 f1特征值都为1，其余9990样本都为0(在高维稀疏的情况下这种情况很常见)，我们都知道这种情况在树模型的时候，很容易优化出含一个使用 f1为分裂节点的树直接将数据划分的很好，但是当测试的时候，却会发现效果很差，因为这个特征只是刚好偶然间跟 y拟合到了这个规律，这也是我们常说的过拟合。但是当时我还是不太懂为什么线性模型就能对这种 case 处理的好？照理说 线性模型在优化之后不也会产生这样一个式子：y = W1*f1 + Wi*fi+....，其中 W1特别大以拟合这十个样本吗，因为反正 f1的值只有0和1，W1过大对其他9990样本不会有任何影响。
>
> 后来思考后发现原因是因为现在的模型普遍都会带着正则项，而 lr 等线性模型的正则项是对权重的惩罚，也就是 W1一旦过大，惩罚就会很大，进一步压缩 W1的值，使他不至于过大，而树模型则不一样，树模型的惩罚项通常为叶子节点数和深度等，而我们都知道，对于上面这种 case，树只需要一个节点就可以完美分割9990和10个样本，惩罚项极其之小，**这也就是为什么在高维稀疏特征的时候，线性模型会比非线性模型好的原因了：带正则化的线性模型比较不容易对稀疏特征过拟合。**

用盖坤的话说，GBDT只是对历史的一个记忆罢了，没有推广性，或者说泛化能力。

但这并不是说对于大规模的离散特征，GBDT和LR的方案不再适用，感兴趣的话大家可以看一下参考文献2和3，这里就不再介绍了。

刚才提到了阿里的盖坤大神，他的团队在2017年提出了两个重要的用于CTR预估的模型，MLR和DIN，之后的系列中，我们会讲解这两种模型的理论和实战！欢迎大家继续关注！

参考文献：

1、Facebook的paper：http://quinonero.net/Publications/predicting-clicks-facebook.pdf
2、http://www.cbdio.com/BigData/2015-08/27/content_3750170.htm
3、https://blog.csdn.net/shine19930820/article/details/71713680
4、https://www.zhihu.com/question/35821566
5、https://github.com/neal668/LightGBM-GBDT-LR/blob/master/GBFT%2BLR_simple.py

推荐阅读：

推荐系统遇上深度学习系列：

推荐系统遇上深度学习(一)--FM模型理论和实践

推荐系统遇上深度学习(二)--FFM模型理论和实践

推荐系统遇上深度学习(三)--DeepFM模型理论和实践

推荐系统遇上深度学习(四)--多值离散特征的embedding解决方案

推荐系统遇上深度学习(五)--Deep&Cross Network模型理论和实践

推荐系统遇上深度学习(六)--PNN模型理论和实践

推荐系统遇上深度学习(七)--NFM模型理论和实践

推荐系统遇上深度学习(八)--AFM模型理论和实践

推荐系统遇上深度学习(九)--评价指标AUC原理及实践

有关作者：

石晓文，中国人民大学信息学院在读研究生，美团外卖算法实习生

简书ID：石晓文的学习日记(https://www.jianshu.com/u/c5df9e229a67)