传统推荐算法(六)Facebook的GBDT+LR模型(2)理论浅析+实战

如雨星空 推荐算法工程师 2019-08-24

前言

点击率预估模型涉及的训练样本一般是上亿级别，样本量大，模型常采用速度较快的LR(logistic regression)。LR虽然是线性模型线性模型，但是在业界广泛使用。为什么呢？虽然模型本身表达能力差，但是可以通过特征工程不断减少问题的非线性结构。又由于模型计算复杂度低，可以吞吐超大规模的特征空间和样本集合，这样就为效果优化打开了空间。同时，他可以学习id化特征，从而减少了特征工程的环节，可以提高特征的实时性[1]。但正因为LR学习能力有限，此时特征工程尤其重要。

在深度学习大行其道之前，一般采用人工或一些一些传统的方法，人工成本高就不说了，传统的方法像FM,FFM，只能挖掘两个特征间的特征交互关系，作用有限。GBDT是解决这个问题的一种不错方案。回顾我们上篇文章所讲的，**GBDT有以下优点**：

- 弱分类器要求不高，树的层数一般较小，小数据可用，扩展到大数据也能方便处理。
- 需要更少的特征工程，比如不用做特征标准化
- 可以处理字段缺失的数据
- 可以自动组合多个特征并且不用关心特征间是否依赖，可以自动处理特征间的交互，不用担心数据是否线性可分
- 可以灵活处理多种类型的异构数据，这是决策树的天然特性
- 损失函数选择灵活，可以选择具有鲁棒性的损失函数，对异常值有一定的鲁棒性

显而易见，GBDT对于处理特征有很多优点。而LR虽然是线性模型，但是Facebook探索出一种将GBDT和LR结合的方案，来预测广告的点击通过率(Click Trough Rate,CTR)的预测问题。结果显示融合方案比单个的GBDT或LR的性能高3%左右。

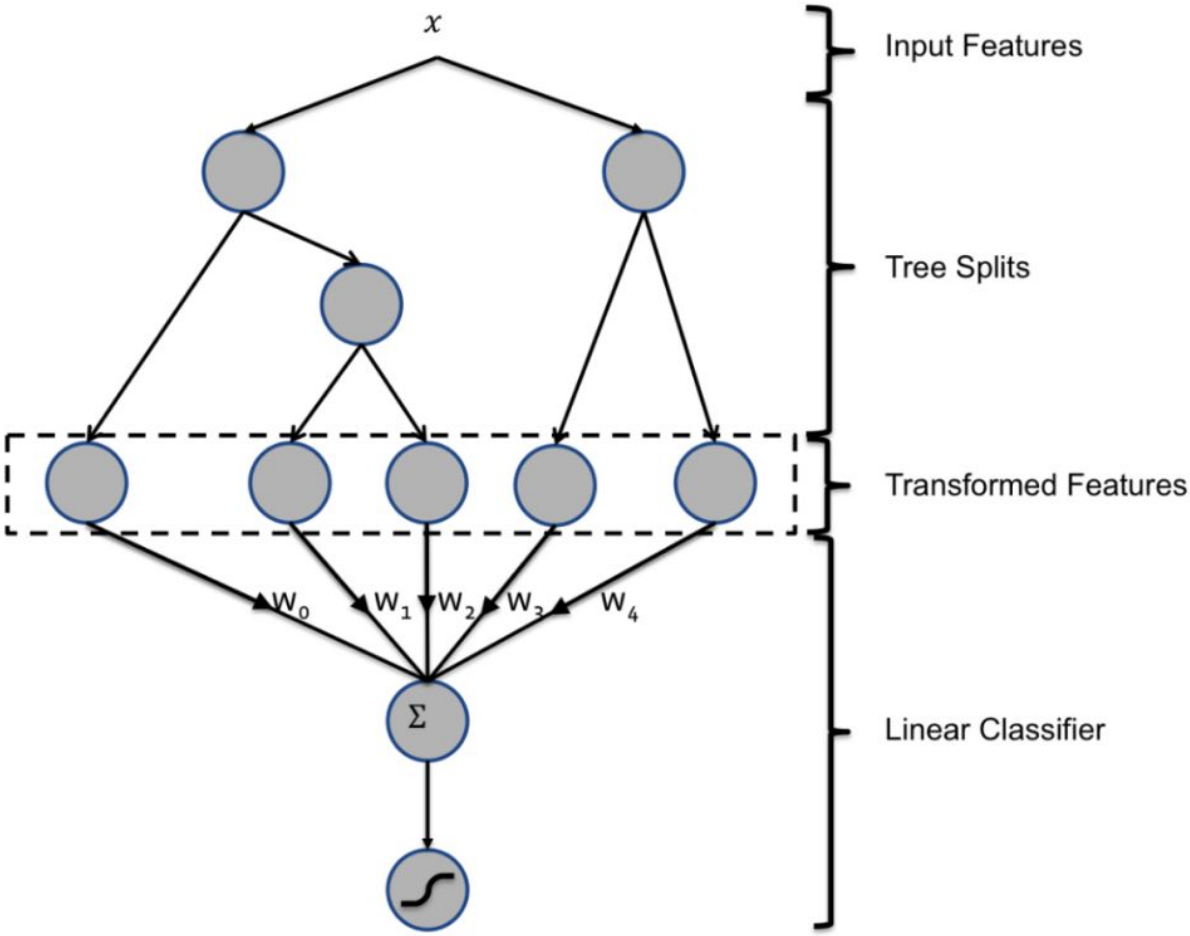
> 论文地址：http://quinonero.net/Publications/predicting-clicks-facebook.pdf
> 代码地址：https://github.com/wyl6/Recommender-Systems-Samples/tree/master/RecSys%20Traditional/DecisionTree/LRGBDT

GBDT+LR模型

首先要说明的是，对点击通过问题，要么点击要么不点击，因此y∈{1,_1}，是个二分类的问题。因此LR+GBDT中的GBDT是上篇文章中所说的L2-Treeboost方案，上篇文章介绍过了这

里不多说。

那么，GBDT与LR是如何结合的呢？看懂论文上的一张图就够了：



如上图所示，这个强学习器由两个弱学习器前向加和组成。假设x输入GBDT后，落到左边回归树的第一个节点，落到右边回归树的第一个节点。则GBDT对样本x的特征进行工程处理得到的转换特征，就可以表示为：(1,0,0)串联（1,0）==》(1,0,0,1,0)。然后将特征输入LR即可进行分类。

反思与总结

GBDT局限性

GBDT+LR这个模型还是有一些局限性的。首先看GBDT的缺点，上篇文章中我们提到过：GBDT有一些缺点：

- GBDT在高维稀疏的数据集上，表现不如支持向量机或者神经网络[2]。
- 训练过程基学习器需要串行训练，只能通过局部并行提高速度。

[3]中凯菜大佬指出，对于高维稀疏的特征，GBDT容易过拟合，表现不理想，甚至LR都比GBDT好，并给了一个例子：

> 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。**

GBDT+LR缺点

关于GBDT+LR的缺点，[4]中屈伟大佬给出了一些看法：

> 1. 离线处理和在线处理都复杂。不同于比赛，在实践中ID类特征还是非常重要的，广告ID可能就有几十万个，深度怎么控制呢？把那么多棵树丢到线上去，然后遍历，拼装特征，想想都难搞。
>
> 2. 要调的参数很多，人生苦短，为什么要搞这么多参数折磨自己。另外再重复一遍：在点击率预估这个问题上，离线效果好往往只能说是模型基本没问题，不能说上线后就效果好。
>
> 3. GDBT+LR本身是想解决特征选择的问题，但现实中也没那么多特征可以用吧？另外没发现点击率预估中如果特征本身没问题，加上去一般都不会降效果吗？
>
> 4. 性能问题怎么解决呢？如果GBDT+LR是不是只能batch方式训练了？如果batch更新速度比FTRL会慢不少，效果又怎么保证呢？想不通。

我们分析模型的优缺点，是为了从中借鉴，也是为了更好地使用模型。没有哪个模型可以解决所有问题，不同模型都有自己的优点和缺点。GBDT+LR这种组合，有局限性，但也提供了一种不错的思路。实际业务中还是根据不同情况选择最合适的模型，扬长补短。

几十行代码的小例子

我们实战一个GBDT对Iris数据集做分类的小例子。改自[4]。参数的使用可以参考官网说明：https://github.com/microsoft/LightGBM/blob/master/docs/Parameters.rst

首先，加载数据：

```
from sklearn.datasets import load_iris
import numpy as np
import lightgbm as lgb
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

## build data
iris = pd.DataFrame(load_iris().data)
iris.columns = ['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']
iris['Species'] = load_iris().target%2
```

由于Iris的labels有三类setosa，versicolor和virginica，而GBDT+LR做CTR是做二分类，因此为了一致，第四行将第三类和第一类合为一类。**然后做训练和测试数据划分：**

```
## train test split
train=iris[0:130]
test=iris[130:]
X_train=train.filter(items=['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm'
X_test=test.filter(items=['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm'])
y_train=train[[train.Species.name]]
y_test=test[[test.Species.name]]
```

样本总共有150个，每类50个，简单划分下得了。**然后构建和训练GBDT模型：**

```
## build lgb model
lgb_train = lgb.Dataset(X_train.as_matrix(),
                        y_train.values.reshape(y_train.shape[0],))
lgb_eval = lgb.Dataset(X_test.as_matrix(),
                       y_test.values.reshape(y_test.shape[0],),
                       reference=lgb_train)
params = {
    'task': 'train',
    'boosting_type': 'gbdt',
    'objective': 'binary',
    'metric': {'binary_logloss'},
    'num_leaves': 16,
    'num_trees': 10,
    'learning_rate': 0.1,
    'feature_fraction': 0.9,
    'bagging_fraction': 0.8,
```

```
    'bagging_freq': 5,
    'verbose': 0
}
gbm = lgb.train(params=params,
                train_set=lgb_train,
                num_boost_round=3000,
                valid_sets=None)
```

使用GBDT做二分类，因此这里我们指定'numtrees'参数.而如果做k(k>2)分类，就要使用'numclass'参数，此时numtrees=numclass*k。**下面GBDT输出转换特征，构建LR训练和测试数据，注意细节：**

```
# build train matrix
num_leaf = 16

y_pred = gbm.predict(X_train,raw_score=False,pred_leaf=True)

transformed_training_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_training_matrix[i][temp] += 1
```

由于我们需要知道每棵树中输出到了那个点上，因此设置参数 `pred_leaf = True`，打印一下看看 `print ( y_pred [ 0 ], y_pred . shape )`：

```
[0 0 0 0 0 0 0 0 0 0 4 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 3 0
 3 0 0 0 0 0 0 0 3 4 3 3 3 0 0 0 0 2 2 3 3 2 3 3 3 1 1 3 3 2 3 1 3 0 3 3 3
 0 2 3 2 3 2 2 2 2 1 1 3 3 0 1 1 3 2 3 0 3 2 0 1 3 3] (130, 100)
```

如果想知道强学习器的预测值y，则需要设置 `raw_score = True '，打印一下看看 print(ypred[0], ypred.shape)`：

```
-9.0297726841214 (130,)
```

预测数据同理：

```
# build test matrix
y_pred = gbm.predict(X_test,pred_leaf=True)
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
```

然后输入到逻辑回归中**分类**：

```
# logistic regression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

label_train = y_train.values.reshape(y_train.shape[0],)
label_test = y_test.values.reshape(y_test.shape[0],)

c = np.array([1,0.5,0.1,0.05,0.01,0.005,0.001])
for t in range(0,len(c)):
    lm = LogisticRegression(penalty='l2',C=c[t]) # logestic model construction
    lm.fit(transformed_training_matrix,y_train.values.reshape(y_train.shape[0],))  # fitt
    y_pred_est = lm.predict(transformed_testing_matrix)   # Give the probabilty on each l
    acc =accuracy_score(label_test, y_pred_est)
    print('Acc of test', acc)
```

100行代码不到，有兴趣可以调调代码：https://github.com/wyl6/Recommender-Systems-Samples/tree/master/RecSys%20Traditional/DecisionTree/LRGBDT

参考

[1] https://www.zhihu.com/question/62109451

[2] http://f.dataguru.cn/thread-935853-1-1.html

[3] https://www.zhihu.com/question/35821566

[4] https://github.com/NearXdu/gbdtlr/blob/master/gbdtlr.py