CTR预估经典模型：GBDT+LR

原创 D.W 斗码小院 2019-08-31

温馨提示：本文是笔者之前发表在http://www.csuldw.com的一篇技术文章，近期在整理公众号，所以就打算将一些质量好点的往这里面迁移，便于手机查阅。
注：欢迎转载，转载请注明出处

在上一篇文章，提到了Facebook 2014年发表的一篇采用GBDT构建特征的论文：Practical lessons from predicting clicks on ads at facebook。为了深入学习GBDT，本文将重点分析这篇文章的思路，即CTR预估经典模型：GBDT+LR，一个曾风靡Kaggle、至今在工业界仍存有余温的传奇模型。同时采用scikit-learn里面的GBDT和LR来完成GBDT+LR的实验。

背景介绍

论文开篇介绍在计算广告领域，Facebook日活用户超过7.5亿，活跃广告超过1百万，这种数据规模对Facebook来说也是一大挑战。在这种情形下，Facebook是怎么做的呢？**引入了一个组合决策树和LR的模型**，该模型比单一的LR或GBDT的效果都要好，不仅将点击率提升了3%，还大大提升了整个系统的性能。除此之外，Facebook还在online learning、data freshness, 学习率等参数上进行了探索。

模型结构

Facebook论文的Section 1给出了一个重要结论：**只要有正确的特征和正确的模型，其他因素对模型结果的影响就非常小**。那么，正确的特征是什么呢？论文对比了两类特征，一类是用户或广告的历史信息特征(historical features)，另一类是contextual features(上下文特征)，相比之下historical features要优于contextual features。正确的模型指的**boosted decision tree + LR**，其中boosted decision tree又相当于对重要的特征做了feature selection。

在Section 3描述了论文的核心模型，整个hybird模型框架示意图如下：

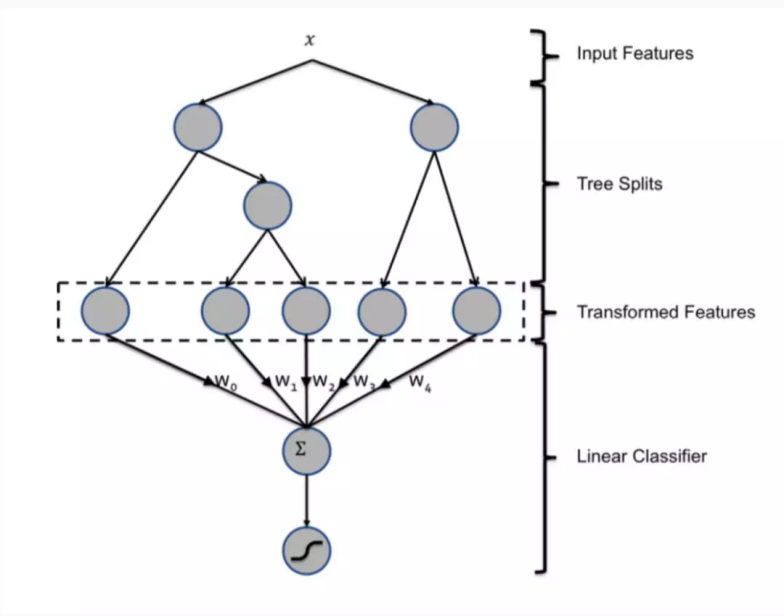

图1：混合模型框架.输入特征经提升树转换，而单颗树的输出又被当作LR的输入.

对于线性分类器，有两种特征转换方式可以提升分类器的精度。

1. 对于连续特征，可以对特征分bin，然后将bin的index作为类别特征，如此线性分类器就可以学习特征的非线性映射，这种方式里，学习有效的bin边界非常重要。
2. 对于类别特征，可以采用笛卡尔积（Cartesian product）枚举出所有的二元特征组合。缺点是得到的特征会包含冗余特征。

为此，基于GBDT的特征转换方法诞生了。

> 基于GBDT的特征转换：将单棵决策树的结果看作是一个类别特征，取值为样本落入在决策树的叶子节点的编号。例如，图1中提升树包含两棵子树，第一棵子树包含3个叶子节点，第二棵树包含2个叶子节点。对于输入样本x（包含多个特征），采用提升决策树（GBDT）进行训练，最终对于第一棵子树上，样本分裂之后落到第二个叶子节点，对于第二棵子树，样本落到了第1个叶子节点，那么通过特征进行转化之后就是 `[0,1,0,1,0]` 。

代码实现

下面通过封装scikit-learn中的GBDT和LR，来实现GBDT+LR的实验。为了代码展示的更美观，这里将GBDT+LR封装到一个类里面 `GradientBoostingWithLR` ，输入数据集的格式与scikit-learn的iris数据格式一致（为了方便，后面也采取iris数据集进行训练和预测）。

GBDT+LR核心方法

```
import numpy as np
from sklearn.ensemble.gradient_boosting import GradientBoostingClassifier
from sklearn.linear_model.logistic import LogisticRegression
from sklearn.metrics.ranking import roc_auc_score
from sklearn.preprocessing.data import OneHotEncoder

class GradientBoostingWithLR(object):
    def __init__(self):
```

```
        self.gbdt_model = None
        self.lr_model = None
        self.gbdt_encoder = None
        self.X_train_leafs = None
        self.X_test_leafs = None
        self.X_trans = None

    def gbdt_train(self, X_train, y_train):
        """定义GBDT模型
        """
        gbdt_model = GradientBoostingClassifier(n_estimators=10,
                                                max_depth=6,
                                                verbose=0,
                                                max_features=0.5)
        # 训练学习
        gbdt_model.fit(X_train, y_train)
        return gbdt_model

    def lr_train(self, X_train, y_train):
        """定义LR模型
        """
        lr_model = LogisticRegression()
        lr_model.fit(X_train, y_train)    # 预测及AUC评测
        return lr_model


    def gbdt_lr_train(self,X_train, y_train,X_test):
        """训练gbdt+lr模型
        """
        self.gbdt_model = self.gbdt_train(X_train, y_train)

        # 使用GBDT的apply方法对原有特征进行编码
        self.X_train_leafs = self.gbdt_model.apply(X_train)[:,:,0]

        # 对特征进行ont-hot编码
        self.gbdt_encoder = OneHotEncoder(categories='auto')
        self.gbdt_encoder.fit(self.X_train_leafs)
        self.X_trans = self.gbdt_encoder.fit_transform(self.X_train_leafs)

        #采用LR进行训练
        self.lr_model = self.lr_train(self.X_trans, y_train)
        return self.lr_model

    def gbdt_lr_pred(self, model, X_test, y_test):
        """预测及AUC评估
        """
        self.X_test_leafs = self.gbdt_model.apply(X_test)[:,:,0]

        (train_rows, cols) =self.X_train_leafs.shape
        X_trans_all = self.gbdt_encoder.fit_transform(np.concatenate((self.X_train_leafs, self.X_tes

        y_pred = model.predict_proba(X_trans_all[train_rows:])[:, 1]
        auc_score = roc_auc_score(y_test, y_pred)
        print('GBDT+LR AUC score: %.5f' % auc_score)
        return auc_score

    def model_assessment(self, model, X_test, y_test, model_name="GBDT"):
        """模型评估
        """
        y_pred = model.predict_proba(X_test)[:,1]
        auc_score = roc_auc_score(y_test, y_pred)
        print("%s AUC score: %.5f" % (model_name,auc_score))
        return auc_score
```

训练与预测

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

def load_data():
    """
    调用sklearn的iris数据集，将多类数据构造成2分类数据，同时切分训练测试数据集
    """
    iris_data = load_iris()
    X = iris_data['data']
    y = iris_data['target'] == 2

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=0)
    return X_train,X_test,y_train, y_test

X_train,X_test,y_train, y_test = load_data()


gblr = GradientBoostingWithLR()
gbdt_lr_model = gblr.gbdt_lr_train(X_train, y_train, X_test)
gblr.model_assessment(gblr.gbdt_model, X_test, y_test)
gblr.gbdt_lr_pred(gbdt_lr_model, X_test, y_test)
```

训练样本落入的叶子节点情况如下（head 10）：

```
print(gblr.X_train_leafs[:10])
```

```
[[ 2.  6.  2.  2.  4.  2.  6.  2.  8.  4.]
 [ 2.  2.  2.  2.  2.  2.  2.  2.  2.  2.]
 [ 7. 10.  7. 10.  9.  6. 12.  6. 12. 10.]
 [ 2.  2.  2.  2.  2.  2.  2.  2.  2.  2.]
 [ 2.  2.  2.  2.  2.  2.  2.  2.  2.  2.]
 [ 2.  2.  2.  2.  2.  2.  2.  2.  2.  2.]
 [ 2.  6.  2.  2.  2.  2.  6.  2.  6.  2.]
 [ 8. 10.  8. 10. 10.  6. 12.  6. 14. 10.]
 [ 2.  2.  2.  2.  2.  2.  2.  2.  2.  2.]
 [ 2.  2.  2.  2.  2.  2.  2.  2.  2.  2.]]
```

采用ont-hot编码之后，结果如下（1条样例）：

```
gblr.X_trans.toarray()[1]
```

```
array([1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 1.,
       0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 1.,
       0., 0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,
       1., 0., 0., 0., 0., 0.])
```

由于数据集较小，最后预测的结果随机性比较大，在参数没有优化的情况下，有时候GBDT的结果反而好于GBDT+LR，所以调参的重要性也是非常大的。

结束语

OK，对于GBDT+LR的介绍到此结束，本文主要是补充一下GBDT的应用以及如何构建GBDT+LR模型（当然你也可以采用其他方式），文中如有纰漏，还望指出。接下来，将介绍boosting模型的下一个进阶算法：XGBoost。

References