实际问题中，可直接用于机器学习模型的特征往往并不多。能否从“混乱”的原始log中挖掘到有用的特征，将会决定机器学习模型效果的好坏。引用下面一句流行的话：

特征决定了所有算法效果的上限，而不同的算法只是离这个上限的距离不同而已。

本文中我将介绍Facebook最近发表的利用GBDT模型构造新特征的方法1。

论文的思想很简单，就是先用已有特征训练GBDT模型，然后利用GBDT模型学习到的树来构造新特征，最后把这些新特征加入原有特征一起训练模型。构造的新特征向量是取值0/1的，向量的每个元素对应于GBDT模型中树的叶子结点。当一个样本点通过某棵树最终落在这棵树的一个叶子结点上，那么在新特征向量中这个叶子结点对应的元素值为1，而这棵树的其他叶子结点对应的元素值为0。新特征向量的长度等于GBDT模型里所有树包含的叶子结点数之和。

举例说明。下面的图中的两棵树是GBDT学习到的，第一棵树有3个叶子结点，而第二棵树有2个叶子节点。对于一个输入样本点x，如果它在第一棵树最后落在其中的第二个叶子结点，而在第二棵树里最后落在其中的第一个叶子结点。那么通过GBDT获得的新特征向量为[0, 1, 0, 1, 0]，其中向量中的前三位对应第一棵树的3个叶子结点，后两位对应第二棵树的2个叶子结点。

那么，GBDT中需要多少棵树能达到效果最好呢？具体数字显然是依赖于你的应用以及你拥有的数据量。一般数据量较少时，树太多会导致过拟合。在作者的应用中，大概500棵左右效果就基本不改进了。另外，作者在建GBDT时也会对每棵树的叶子结点数做约束——不多于12个叶子结点。

下面是这种方法在我们世纪佳缘的一个概率预测问题上的实际效果。我们只使用了30棵树。第一个图是只使用原始特征的结果，第二个图是原始特征加GBDT新特征的结果。图中横坐标表示预测概率值，纵坐标表示真实概率值。所以预测的点越靠近这条参考线越好。显然，使用了GBDT构造的新特征后，模型的预测效果好不少。

对了，已经有人利用这种方法赢得了Kaggle一个CTR预估比赛的冠军，代码可见https://github.com/guestwalk/kaggle-2014-criteo，里面有这种方法的具体实现。

以下是xgboost生成新特征的一个实现，需要xgb0.6版本及以上

示例代码使用的是sklearn的apply()接口，也可以使用xgboost原生接口，new\_feature= bst.predict(X\_test, pred\_leaf=True)即可得到一个(nsample, ntrees) 的结果矩阵

# -\*- coding: utf-8 -\*-

"""

Created on Mon Jul 3 21:37:30 2017

@author: bryan

"""

from sklearn.model\_selection import train\_test\_split

from sklearn import metrics

from xgboost.sklearn import XGBClassifier

import pandas as pd

import numpy as np

def mergeToOne(X,X2):

X3=[]

for i in range(X.shape[0]):

tmp=np.array([list(X.iloc[i]),list(X2[i])])

X3.append(list(np.hstack(tmp)))

X3=np.array(X3)

return X3

data=pd.read\_csv("e:\data\wine.csv")

#打乱数据

data=data.sample(len(data))

y=data.label

X=data.drop("label",axis=1)

#划分训练集测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.1, random\_state=0)##test\_size测试集合所占比例

##X\_train\_1用于生成模型 X\_train\_2用于和新特征组成新训练集合

X\_train\_1, X\_train\_2, y\_train\_1, y\_train\_2 = train\_test\_split(X\_train, y\_train, test\_size=0.6, random\_state=0)

clf = XGBClassifier(

learning\_rate =0.2, #默认0.3

n\_estimators=200, #树的个数

max\_depth=8,

min\_child\_weight=10,

gamma=0.5,

subsample=0.75,

colsample\_bytree=0.75,

objective= 'binary:logistic', #逻辑回归损失函数

nthread=8, #cpu线程数

scale\_pos\_weight=1,

reg\_alpha=1e-05,

reg\_lambda=10,

seed=1024) #随机种子

clf.fit(X\_train\_1, y\_train\_1)

new\_feature= clf.apply(X\_train\_2)

X\_train\_new2=mergeToOne(X\_train\_2,new\_feature)

new\_feature\_test= clf.apply(X\_test)

X\_test\_new=mergeToOne(X\_test,new\_feature\_test)

model = XGBClassifier(

learning\_rate =0.05, #默认0.3

n\_estimators=300, #树的个数

max\_depth=7,

min\_child\_weight=1,

gamma=0.5,

subsample=0.8,

colsample\_bytree=0.8,

objective= 'binary:logistic', #逻辑回归损失函数

nthread=8, #cpu线程数

scale\_pos\_weight=1,

reg\_alpha=1e-05,

reg\_lambda=1,

seed=1024) #随机种子

model.fit(X\_train\_2, y\_train\_2)

y\_pre= model.predict(X\_test)

y\_pro= model.predict\_proba(X\_test)[:,1]

print("AUC Score :",(metrics.roc\_auc\_score(y\_test, y\_pro)))

print("Accuracy :",(metrics.accuracy\_score(y\_test, y\_pre)))

model = XGBClassifier(

learning\_rate =0.05, #默认0.3

n\_estimators=300, #树的个数

max\_depth=7,

min\_child\_weight=1,

gamma=0.5,

subsample=0.8,

colsample\_bytree=0.8,

objective= 'binary:logistic', #逻辑回归损失函数

nthread=8, #cpu线程数

scale\_pos\_weight=1,

reg\_alpha=1e-05,

reg\_lambda=1,

seed=1024) #随机种子

model.fit(X\_train\_new2, y\_train\_2)

y\_pre= model.predict(X\_test\_new)

y\_pro= model.predict\_proba(X\_test\_new)[:,1]

print("AUC Score :",(metrics.roc\_auc\_score(y\_test, y\_pro)))

print("Accuracy :",(metrics.accuracy\_score(y\_test, y\_pre)))