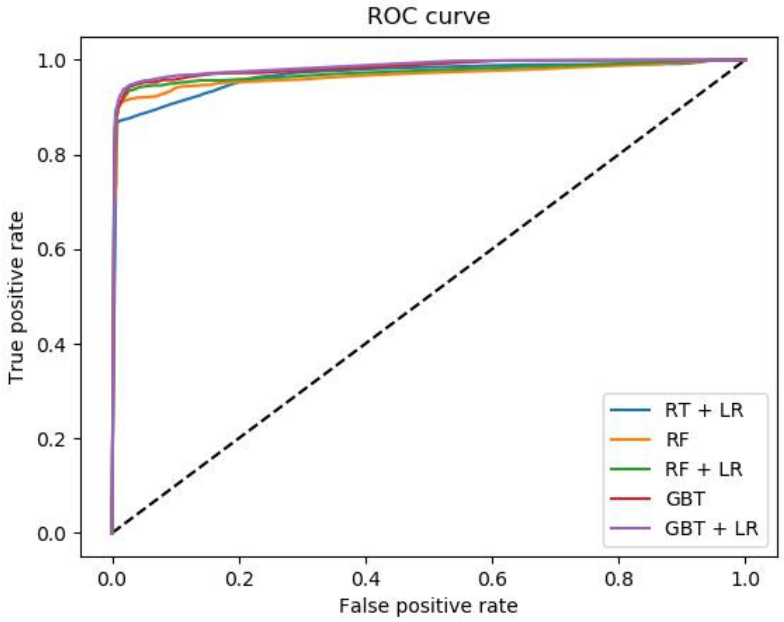
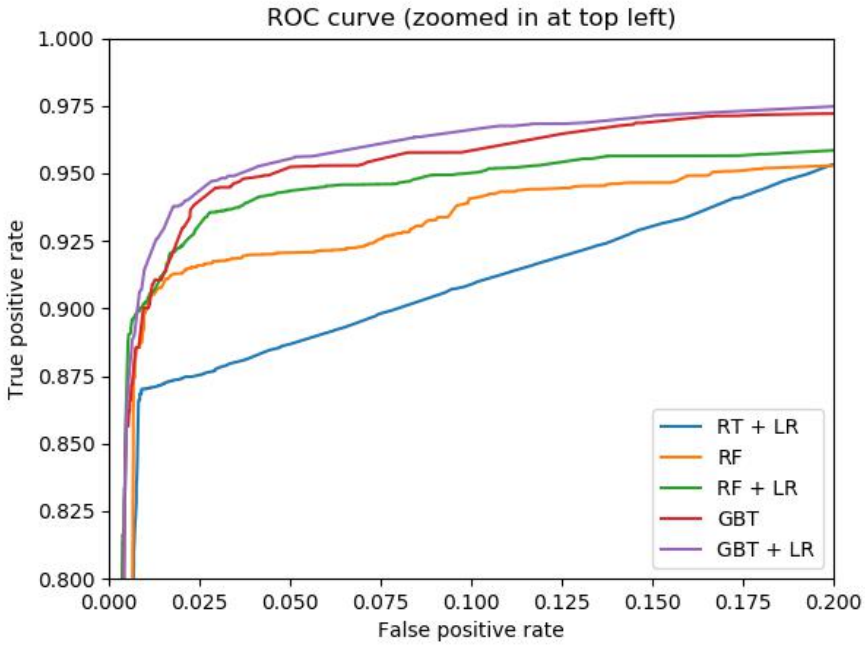
# GBDT+LR

如果基础模型的效果差强人意，适当的改进往往可以提升模型学习能力，而基础模型的组合就是一种简单有效的常用方式。GBDT+LR 模型作为一种混合模型，既带有 GBDT 树模型的天然特征处理属性，又不失 LR 广义线性模型方便易用的特点，犹如男女搭配，各显其长。



将您的特征转换为更高维度的稀疏空间。然后训练这些特征的线性模型。首先在训练集上使用一组树模型（完全随机的树模型 RT，随机森林 RF 或梯度提升的树 GBDT）。然后，在集合中的每个树的每个叶子节点被分配在新特征空间中的固定的任意特征索引。然后以独热的方式编码这些叶索引。

**每个样本都经过整体的每棵树的决定，并以每棵树的一片叶子结束。**通过将这些叶的特征值设置为 1 并将其他特征值设置为 0 来对样本进行编码。

然后，所得到的 transformer 学习数据的监督的，稀疏的，高维的分类嵌入。

http://scikit-learn.org/stable/modules/generated/sklearn.metrics.auc.html#sklearn.metrics.auc

http://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html#sklearn.metrics.roc_auc_score

http://scikit-learn.org/stable/auto_examples/ensemble/plot_feature_transformation.html 代码实现

# 1、算法背景

2014 年 Facebook 发表了一篇介绍将 GBDT+LR 模型用于其广告推荐系统的论文，之后，无论是 Kaggle 竞赛还是淘宝商品推荐，都有借鉴该论文中的 GBDT+LR 模型组合思想，即通过 GBDT 来发掘有区分度的特征和组合特征，来代替人工组合特征。

对于支撑互联网半壁江山的广告收入，推荐系统和 CTR 预估于其技术框架中占据重要地位，而 LR 模型则是其中最为常用的模型。

LR 模型有以下特点：

- 计算复杂度低；
- 易于并行化处理；

易于得到离散化目标值 0 或 1，利用 sigmoid 函数将传统线性模型的输出值映射到(0,1)区间；

学习能力限于线性特征，需要提前进行大量的特征工程得到有效的特征及特征组合。

输入 LR 模型的特征很重要，但是特征组合不能直接通过特征笛卡尔积获取，只能依靠人工经验。故而如何自动化进行特征工程，规范化 LR 模型使用流程是一个值得研究的问题。

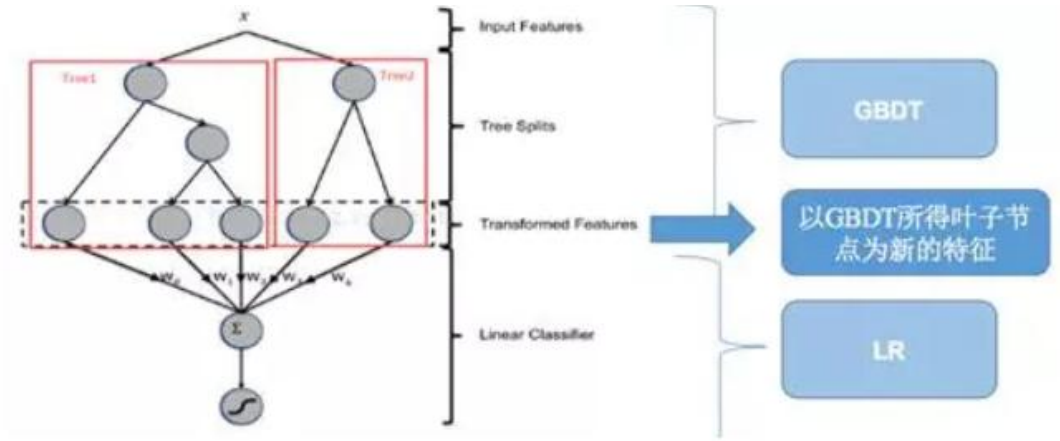
GBDT 作为一种常用的树模型，可天然地对原始特征进行特征划分、特征组合和特征选择，并得到高阶特征属性和非线性映射。从而可将 GBDT 模型抽象为一个特征处理器，通过 GBDT 分析原始特征获取到更利于 LR 分析的新特征。这也正是 GBDT+LR 模型的核心思想——利用 GBDT 构造的新特征来训练 LR 模型。

# 2、算法原理及实现

前面简单介绍了 GBDT+LR 模型的产生背景和核心思想，接下来将会更为详细地描述 GBDT+LR 模型的算法组合思想和简单实现流程。

## 2.1、算法组合——stacking

stacking 方法有些类似于农业中的嫁接，通过 stacking 方法组合的模型亦类似于嫁接植物，例如，解决了人类吃饭问题的杂交水稻。




如上图所示，GBDT 算法的图示部分形如一棵倒过来的树，其根部即代表训练 GBDT 算法的原始数据集，经过树算法对原始数据的切分，可得到代表不同新特征的叶子节点。

再将 GBDT 所得的叶子节点输入 LR 算法，经过线性分析和 sigmoid 映射，即可得到模型分类结果。

以上的模型组合方式就是 stacking 方法，即将学习层模型对原始数据所得的预测结果作为新的特征集，并输入给输出层模型得到分类结果。Facebook 论文中的 GBDT+LR 模型就采用了 GBDT 算法作为学习层，以 LR 算法为输出层。

## 2.2、算法流程& 代码简单实现


在这一部分中，GBDT+LR 算法的代码实现语言为 python，使用了 sklearn 包中的 GradientBoostingClassifier 和 LogisticRegression 函数作为 GBDT 模型和 LR 模型。

将训练集记为 (X,y) ，其中 X 为原始特征，y 为目标变量。

## 数据预处理

对变量取值中的中英文字符、缺失值和正负无穷值进行处理。

## 数据集划分

为了降低过拟合的风险，将训练集中的数据划分为两部分，一部分数据用于训练 GBDT 模型，另一部分数据通过训练好的 GBDT 模型得到新特征以训练 LR 模型。

```
fromsklearn.model import train_test_split
X_gbdt,X_lr,y_gbdt,y_lr= train_test_split(X,y,test_size=0.5)
```

## GBDT 特征转化

首先，通过 sklearn 中的 GradientBoostingClassifier 得到 GBDT 模型，然后使用 GBDT 模型的 fit 方法训练模型，最后使用 GBDT 模型的 apply 方法得到新特征。

```
fromsklearn.ensemble import GradientBoostingClassifier
       gbdt = GradientBoostingClassifier()
       gbdt.fit(X_gbdt,y_gbdt)
       leaves = gbdt.apply(X_lr)[:,:,0]
```

- 

## 特征独热化

使用 sklearn.preprocessing 中的 OneHotEncoder 将 GBDT 所得特征独热化。

```
fromsklearn.preprocessing import OneHotEncoder
```

```
featutes_trans =OneHotEncoder.fit_transform(leaves)
```

- LR 进行分类

用经过离散化处理的新特征训练 LR 模型并得到预测结果。

```
fromsklearn.linear_model import LogisticRegression
lr= LogisticRegression()
    lr.fit(features_trans,y_lr)
    lr.predict(features_trans)
    lr.predict_proba(features_trans)[:,1]
```

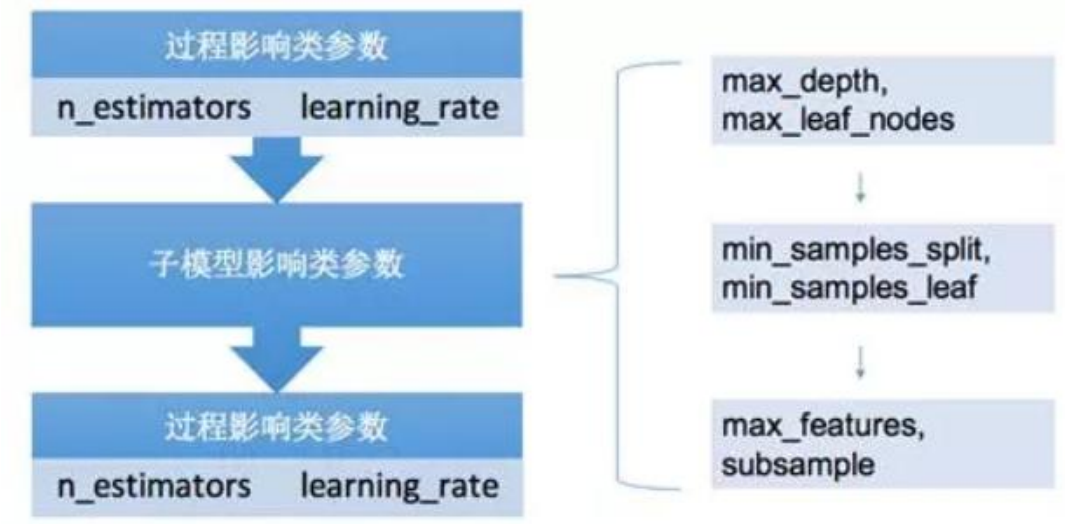
## 2.3、调参方法简述

构建了模型框架后，模型中的函数参数调整也是必不可少的。对模型参数的适当调整，往往可以有效提升模型的效果。

由于 GBDT+LR 模型无法整体使用 GridSearchCV 函数，所以调参时

使用 sklearn.cross_validation 中的 StratifiedKFold 方法，将数据集进行 k 折交叉切分，然后以 auc 值为模型评估指标，对混合模型进行调参。

调参时的重点为 GradientBoostingClassifier 函数，可用如下图所示的调参顺序进行调参。




其中，n_estimators 和 learning_rate 应该联合调参。

## 2.4、模型效果展示

在介绍了 GBDT+LR 模型的原理和实现流程之后，我们以一个 1.5 万条的数据样本为例，来比较直观地认识一下模型效果。

我们分别使用 LR 模型和 GBDT+LR 模型对样本数据集进行学习，通过模型所得的 auc 值和 ks 值，来评估和比较模型的效果。

| 模型 | AUC_train | AUC_test | delta_AUC | ks_train | ks_test | delat_ks |
|---|---|---|---|---|---|---|
| LR | 0.712 | 0.702 | 1.0% | 0.331 | 0.318 | 1.3% |
| GBDT+LR | 0.894 | 0.873 | 1.9% | 0.618 | 0.592 | 2.6% |

如上图所示，可知 GBDT+LR 模型的效果要更好一些，即 GBDT 所得的新特征的确更适合 LR 模型的分析。

# 3、算法引申

前面的内容描述了 Facebook 论文中 GBDT+LR 混合模型的算法原理并附有简单实现代码。然而，模型并不可孤立地比较好坏，模型的应用也要和应用场景及数据质量互相照应。

这一部分将会简单提供一些 GBDT+LR 混合模型的引申思路，希望对大家实际使用时有所裨益。

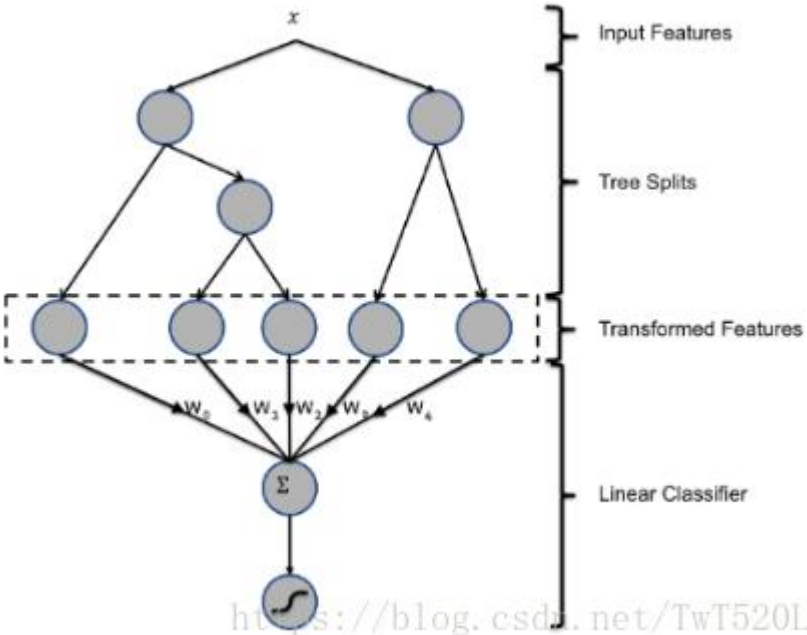
- 用 FFM 模型替代 LR 模型；
- 直接将 GBDT 所得特征输入 FFM 模型；
- 用 XGBoost 模型替代 GBDT 模型；
- 将 stacking 模型学习层中的 GBDT 交叉检验；
- GBDT 和 LR 模型使用 model fusion，而不是 stacking

# 【实战】GBDT+LR 算法进行特征扩增

## 0.简介

CTR 估计也就是广告点击率预估，计算广告训练与平滑思想说明了是用 LR 算法对于预测的有效性。LR（Logistic Regression）是广义线性模型，与传统线性模型相比，LR 通过 Logit 变换将函数值映射到 0~1 区间，映射后的函数就是 CTR 的预估值。LR 模型十分适合并行化，因此对于大数据的训练十分有效。但是对于线性模型而言，学习能力是有限的，因此需要大量的特征工程预先分析出有效的特征或者是特征组合，从而去间接的增强 LR 的非线性学习能力。

特征组合，是通过特征的一些线性叠加或者非线性叠加得到一个新的特征，可以有效的提高分类效果。常见的特征组合方式有笛卡尔积方式。为了降低人工组合特征的工作量，FaceBook 提出了一个自动特征提取的方式 GBDT+LR。




GBDT 是梯度提升决策树，首先会构造一个决策树，首先在已有的模型和实际样本输出的残差上再构造一颗决策树，不断地进行迭代。每一次迭代都会产生一个增益较大的分类特征，因此 GBDT 树有多少个叶子节点，得到的特征空间就有多大，并将该特征作为 LR 模型的输入。

# 1. 核心问题

（1）建树采用 ensemble 决策树？

一棵树的区分性是具有一定的限制的，但是多棵树可以获取多个具有区分度的特征组合，而且 GBDT 的每一棵树都会学习前面的树的不足。

（2）建树算法为什么采用 GBDT 而不是 RF?

对于 GBDT 而言，前面的树，特征分裂主要体现在对多数样本的具有区分度的特征；后面的树，主要体现的是经过前面 n 棵树，残差依然比较大的少数样本。优先选用在整体上具有区分度的特征，再选用针对少数样本有区分度的特征。

# 2. 代码实现

导入包

```
import numpy as np

import random
```

```
import xgboost as xgb
import matplotlib
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from xgboost.sklearn import XGBClassifier
```

生成随机数据

```
np.random.seed(10)
X, Y = make_classification(n_samples=1000, n_features=30)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=233,
test_size=0.5)
X_train, X_train_lr, Y_train, Y_train_lr = train_test_split(X_train,
Y_train, random_state=233, test_size=0.2)
```

### RandomForest + LogisticRegression

```
def RandomForestLR():
    RF = RandomForestClassifier(n_estimators=100, max_depth=4)
    RF.fit(X_train, Y_train)
    OHE = OneHotEncoder()
    OHE.fit(RF.apply(X_train))
    LR = LogisticRegression()
    LR.fit(OHE.transform(RF.apply(X_train_lr)), Y_train_lr)
    Y_pred = LR.predict_proba(OHE.transform(RF.apply(X_test)))[:, 1]
    fpr, tpr, _ = roc_curve(Y_test, Y_pred)
    auc = roc_auc_score(Y_test, Y_pred)
    print('RandomForest + LogisticRegression: ', auc)
    return fpr, tpr
```

### Xgboost + LogisticRegression

```
def XGBoostLR():
    XGB = xgb.XGBClassifier(nthread=4, learning_rate=0.08,
n_estimators=100, colsample_bytree=0.5)
    XGB.fit(X_train, Y_train)
    OHE = OneHotEncoder()
    OHE.fit(XGB.apply(X_train))
    LR = LogisticRegression(n_jobs=4, C=0.1, penalty='l1')
    LR.fit(OHE.transform(XGB.apply(X_train_lr)), Y_train_lr)
```

```
    Y_pred = LR.predict_proba(OHE.transform(XGB.apply(X_test)))[:, 1]
    fpr, tpr, _ = roc_curve(Y_test, Y_pred)
    auc = roc_auc_score(Y_test, Y_pred)
    print('XGBoost + LogisticRegression: ', auc)
    return fpr, tpr
```

### GradientBoosting + LogisticRegression

```
def GBDTLR():
    GBDT = GradientBoostingClassifier(n_estimators=10)
    GBDT.fit(X_train, Y_train)
    OHE = OneHotEncoder()
    OHE.fit(GBDT.apply(X_train)[:, :, 0])
    LR = LogisticRegression()
    LR.fit(OHE.transform(GBDT.apply(X_train_lr)[:, :, 0]), Y_train_lr)
    Y_pred = LR.predict_proba(OHE.transform(GBDT.apply(X_test)[:, :,
0]))[:, 1]
    fpr, tpr, _ = roc_curve(Y_test, Y_pred)
    auc = roc_auc_score(Y_test, Y_pred)
    print('GradientBoosting + LogisticRegression: ', auc)
    return fpr, tpr
```

### LogisticRegression

```
def LR():
    LR = LogisticRegression(n_jobs=4, C=0.1, penalty='l1')
    LR.fit(X_train, Y_train)
    Y_pred = LR.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(Y_test, Y_pred)
    auc = roc_auc_score(Y_test, Y_pred)
    print('LogisticRegression: ', auc)
    return fpr, tpr
```

### XGBoost

```
def XGBoost():
    XGB = xgb.XGBClassifier(nthread=4, learning_rate=0.08,
n_estimators=100, colsample_bytree=0.5)
    XGB.fit(X_train, Y_train)
    Y_pred = XGB.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(Y_test, Y_pred)
    auc = roc_auc_score(Y_test, Y_pred)
    print('XGBoost: ', auc)
    return fpr, tpr
```

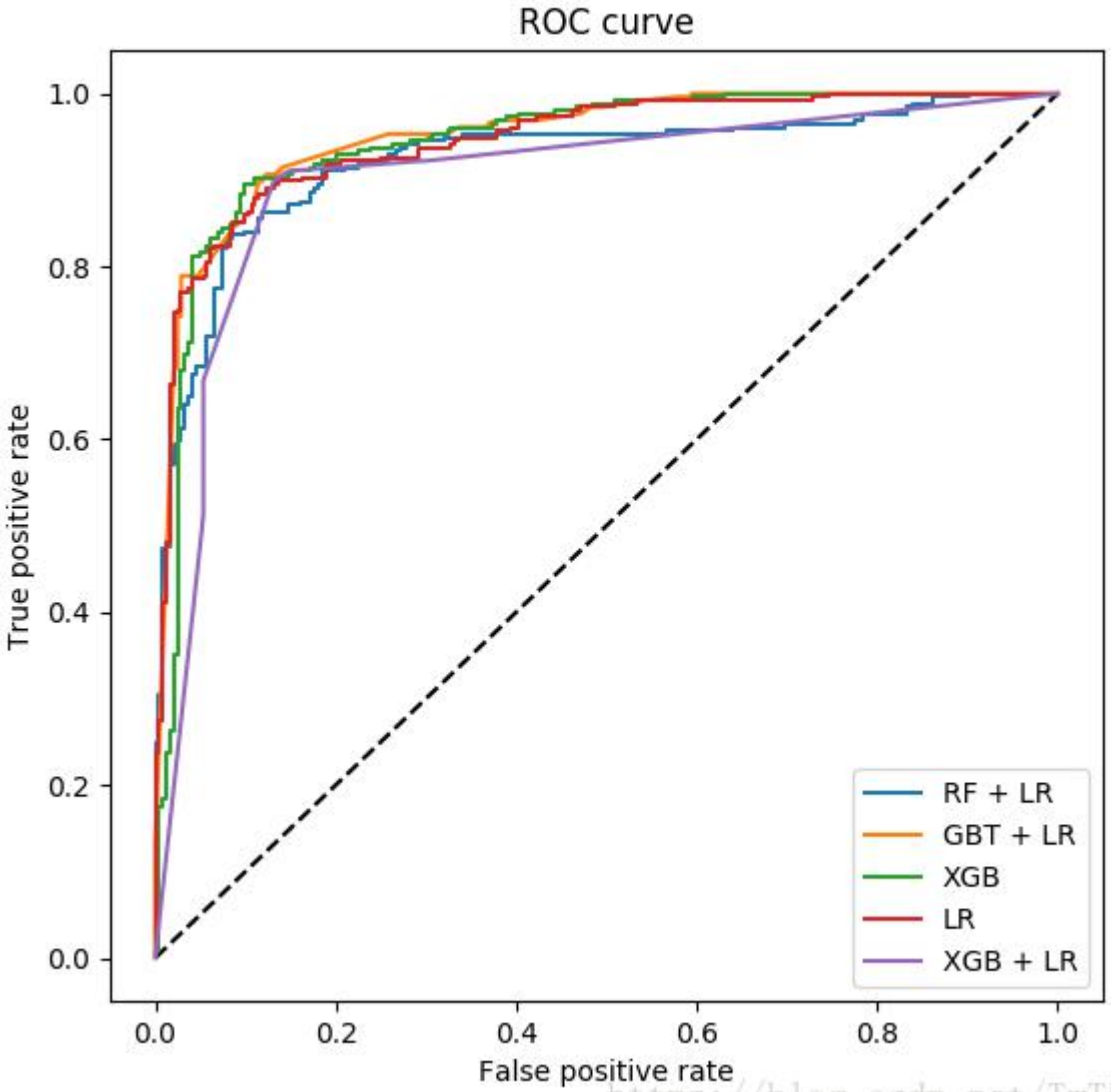
调用并绘制图像

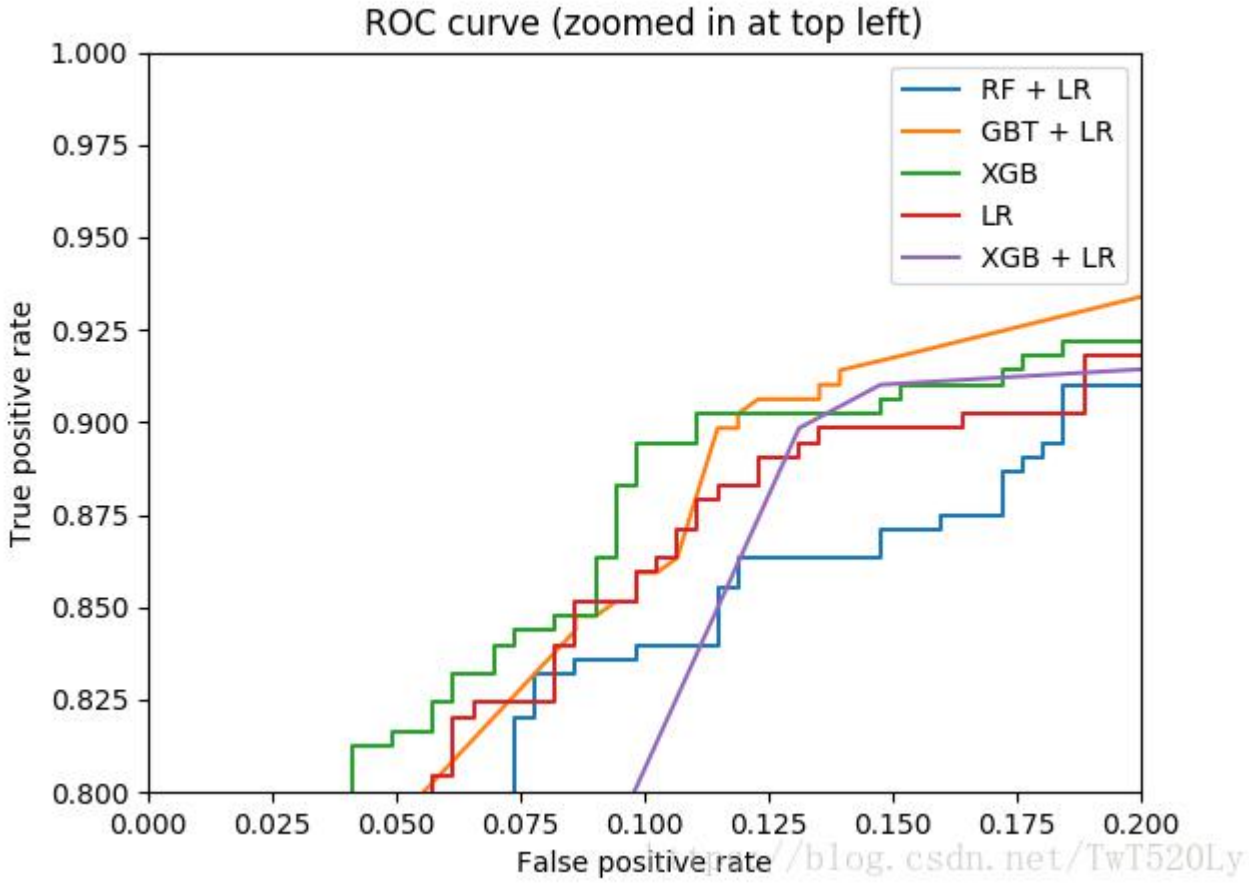
```
if __name__ == '__main__':
```

```
fpr_xgb_lr, tpr_xgb_lr = XGBoostLR()
fpr_xgb, tpr_xgb = XGBoost()
fpr_lr, tpr_lr = LR()
fpr_rf_lr, tpr_rf_lr = RandomForestLR()
fpr_gbdt_lr, tpr_gbdt_lr = GBDTLR()

plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_rf_lr, tpr_rf_lr, label='RF + LR')
plt.plot(fpr_gbdt_lr, tpr_gbdt_lr, label='GBT + LR')
plt.plot(fpr_xgb, tpr_xgb, label='XGB')
plt.plot(fpr_lr, tpr_lr, label='LR')
plt.plot(fpr_xgb_lr, tpr_xgb_lr, label='XGB + LR')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()

plt.figure(2)
plt.xlim(0, 0.2)
plt.ylim(0.8, 1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_rf_lr, tpr_rf_lr, label='RF + LR')
plt.plot(fpr_gbdt_lr, tpr_gbdt_lr, label='GBT + LR')
plt.plot(fpr_xgb, tpr_xgb, label='XGB')
plt.plot(fpr_lr, tpr_lr, label='LR')
plt.plot(fpr_xgb_lr, tpr_xgb_lr, label='XGB + LR')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve (zoomed in at top left)')
plt.legend(loc='best')
plt.show()
```






参考文献：

https://mp.weixin.qq.com/s?__biz=MzI1ODM5MTI4Nw%3D%3D&chksm=ea09a6badd7e2fac05f9886746bd717bc7e53503906728337b72cd1b95cd2faa4e186e79b9cd&idx=1&mid=2247486242&scene=21&sn=3723bc28c36e0c779bb20aa3f1c92b23

https://blog.csdn.net/lilyth_lilyth/article/details/48032119

https://blog.csdn.net/asdfghjkl1993/article/details/78606268

https://blog.csdn.net/TwT520Ly/article/details/79769705