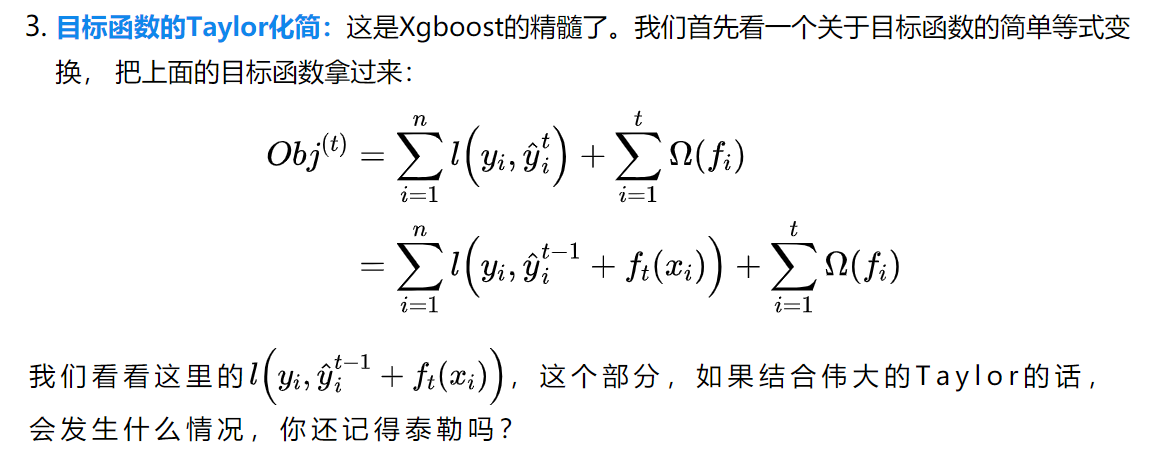
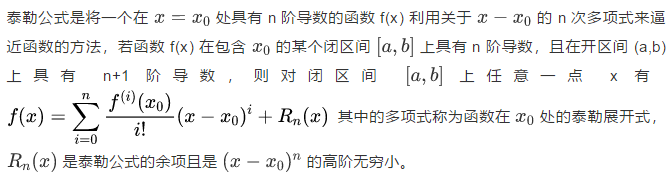
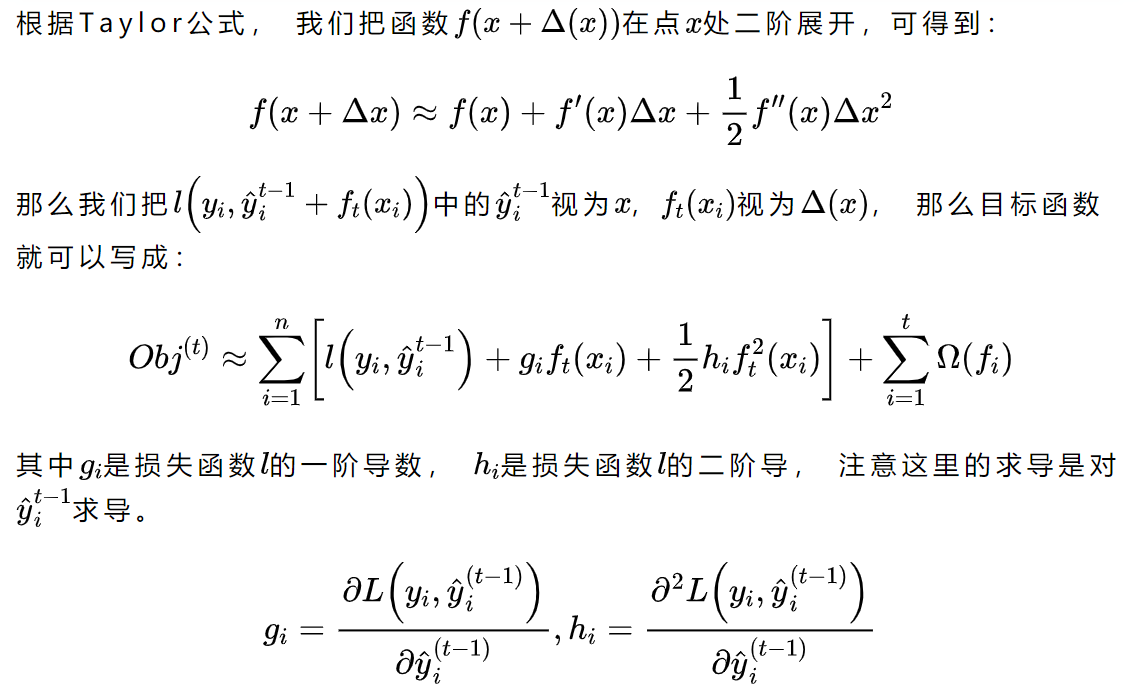
Gbdt，xgboost

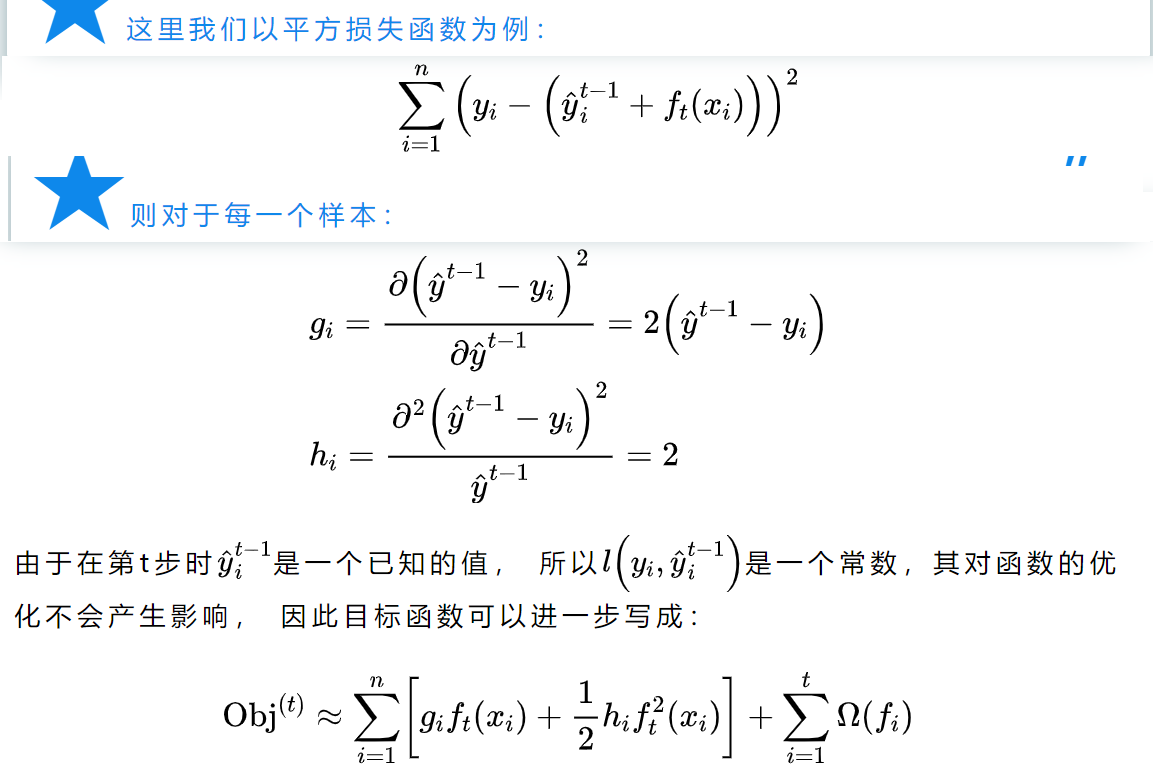
# 1.陈天奇的xgboost ppt;

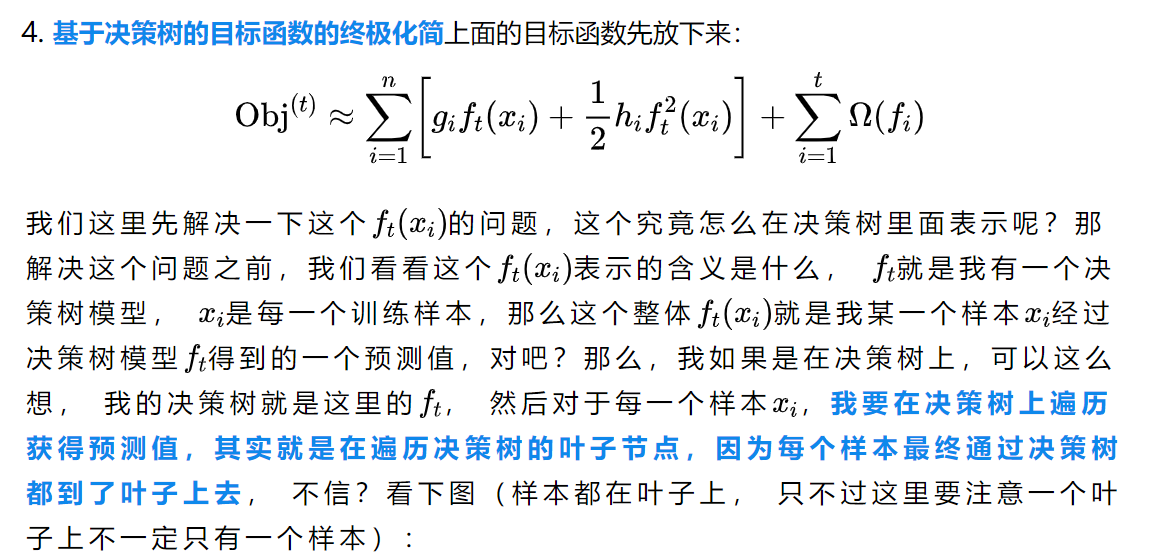
<https://mp.weixin.qq.com/s/NC9CwR4cfDUJ26WpHsvkPQ>

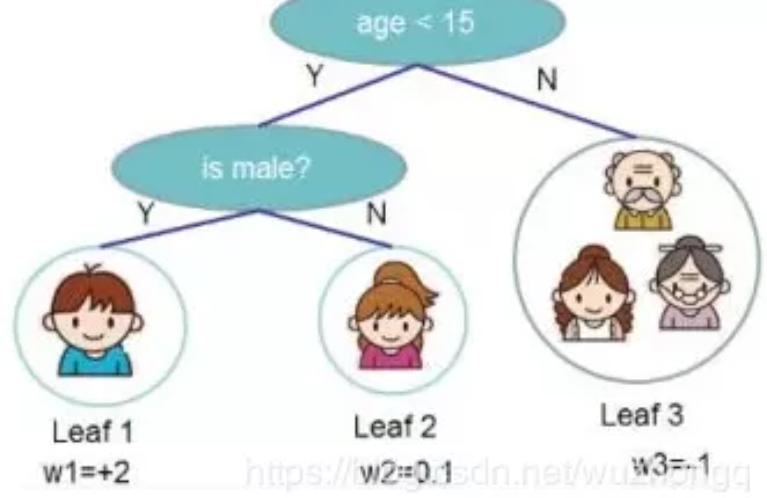


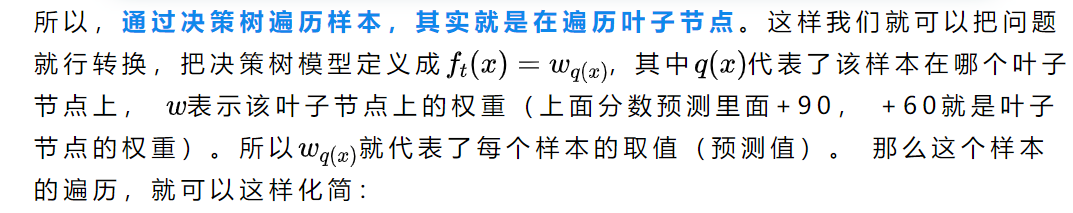


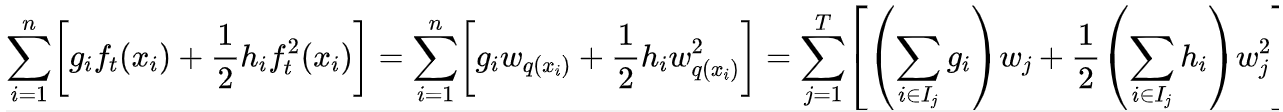


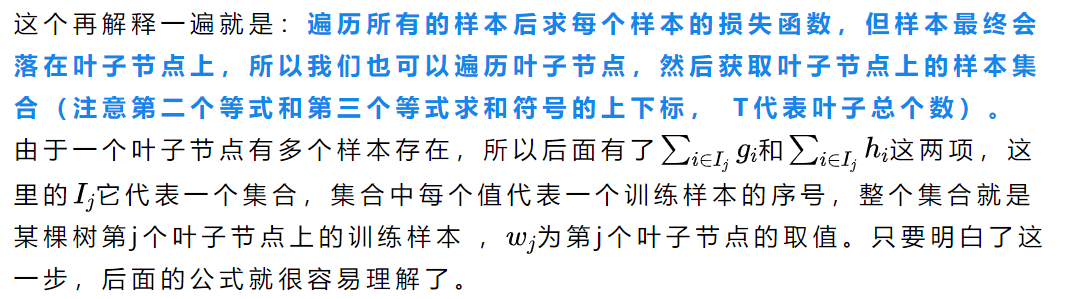


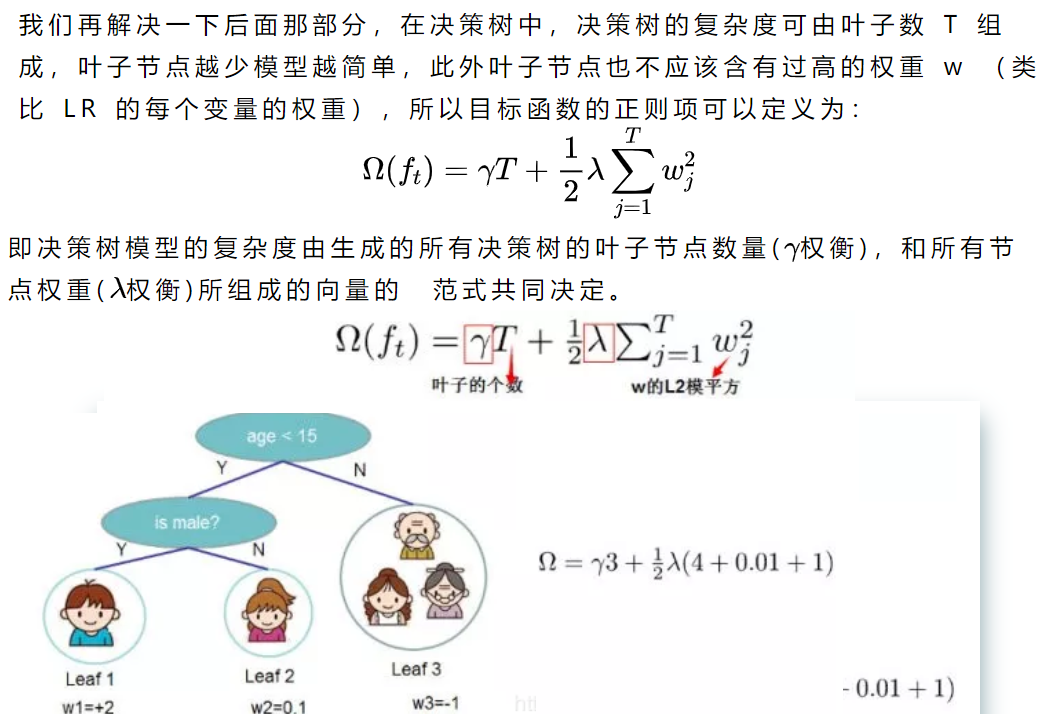


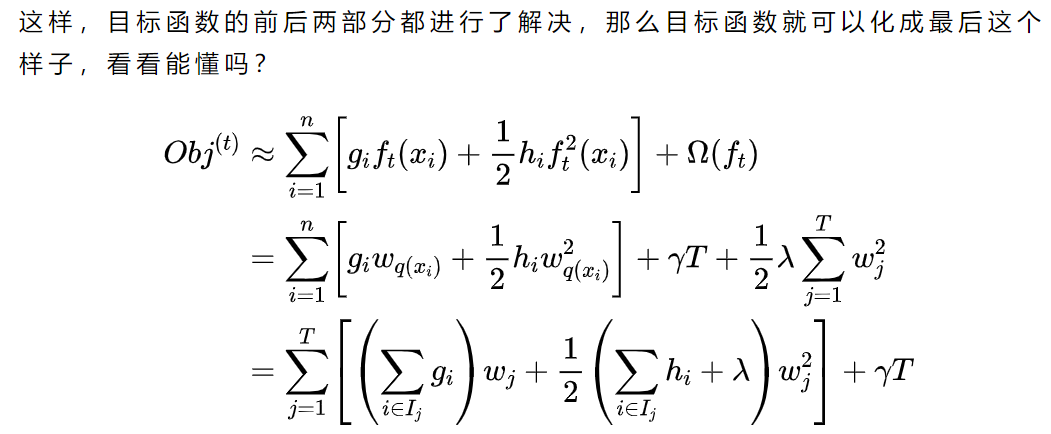


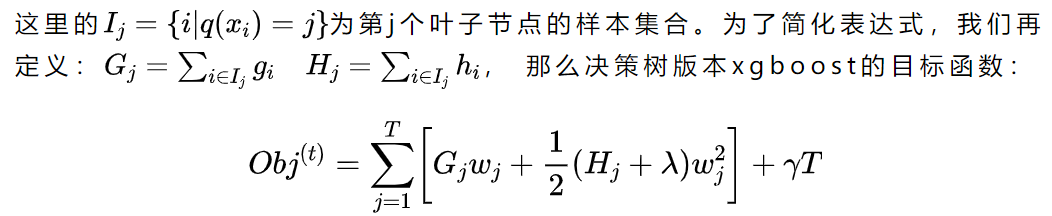


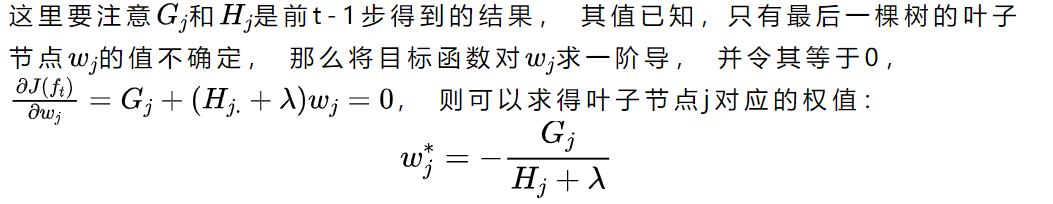


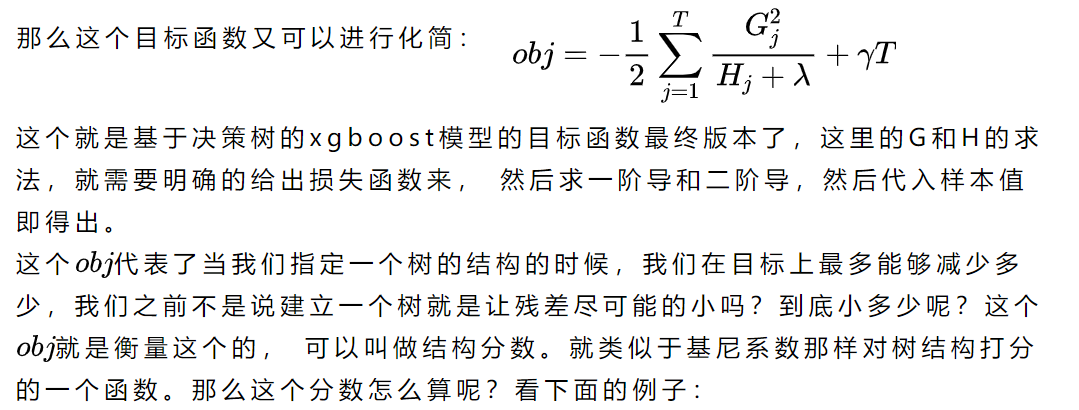


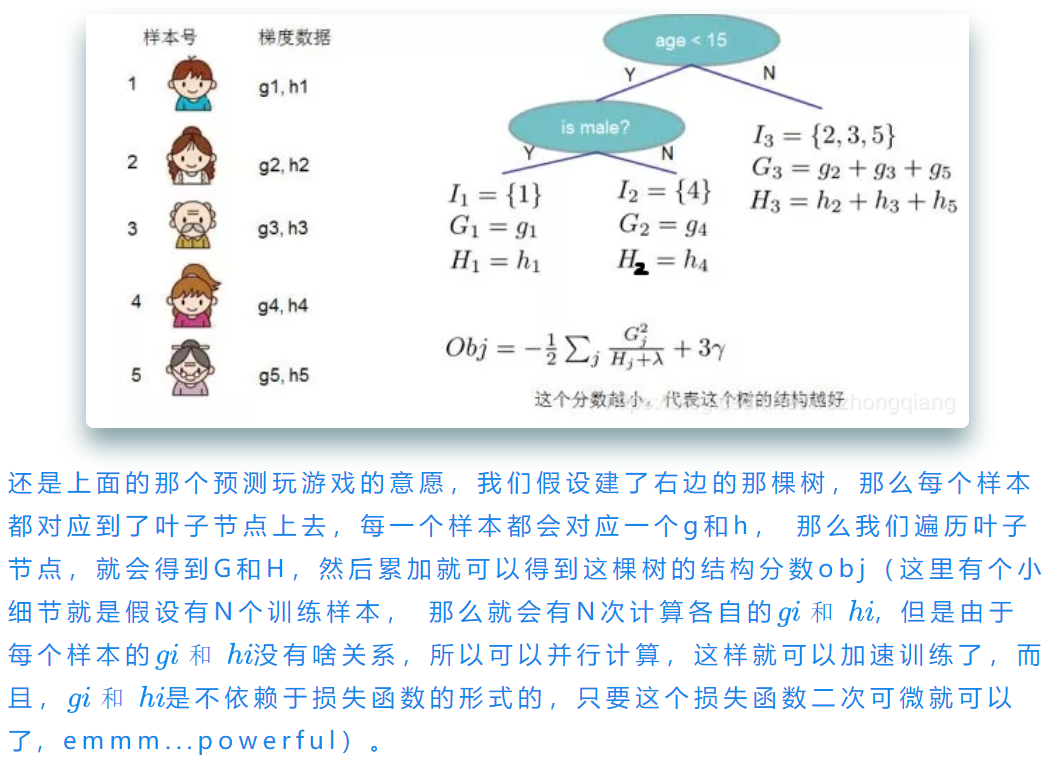






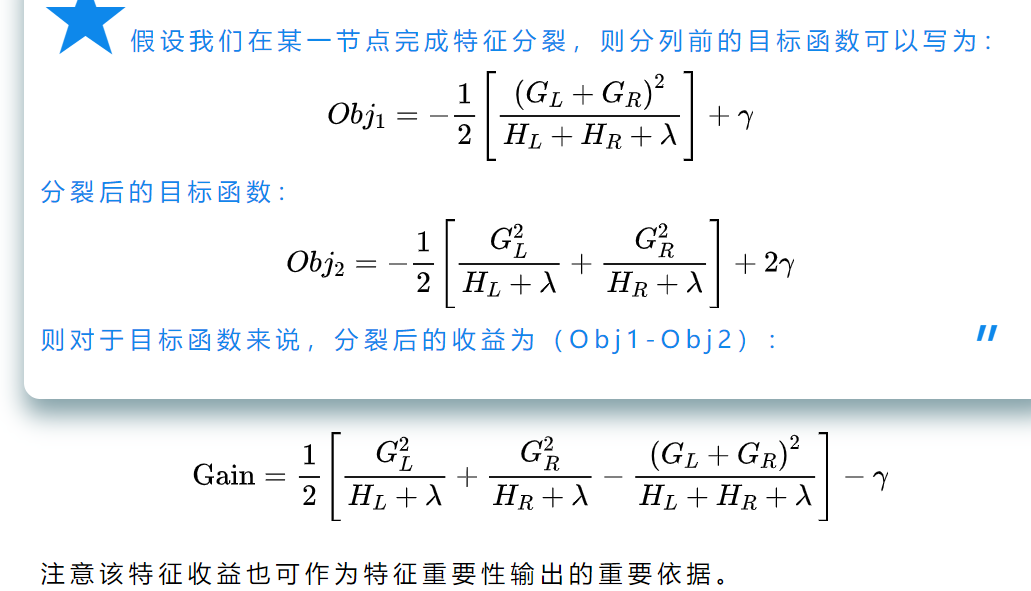






上面是可以判断出来一棵树究竟好不好，那么建立树的时候应该怎么建立呢？一棵树的结构近乎无限多，总不能一个一个去测算它们的好坏程度，然后再取最好的吧（这是个NP问题）。所以，我们仍然需要采取一点策略，这就是逐步学习出最佳的树结构。这与我们将K棵树的模型分解成一棵一棵树来学习是一个道理，只不过从一棵一棵树变成了一层一层节点而已。这叫什么？emmm, 贪心（找到每一步最优的分裂结果）！xgboost采用二叉树， 开始的时候， 全部样本在一个叶子节点上， 然后叶子节点不断通过二分裂，逐渐生成一棵树。  
  
那么在叶子节点分裂成树的过程中最关键的一个问题就是应该在哪个特征的哪个点上进行分裂，也就是寻找最优切分点的过程。

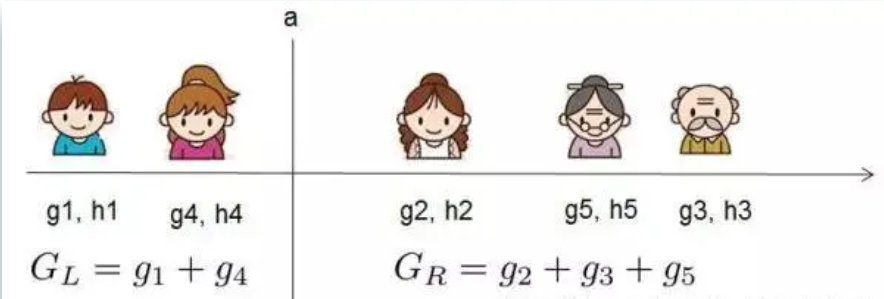
**最优切分点划分算法及优化策略**在决策树的生长过程中，一个非常关键的问题是如何找到节点的最优切分点， 我们学过了决策树的建树过程，那么我们知道ID3也好，C4.5或者是CART，它们寻找最优切分点的时候都有一个计算收益的东西，分别是信息增益，信息增益比和基尼系数。而xgboost这里的切分， 其实也有一个类似于这三个的东西来计算每个特征点上分裂之后的收益。

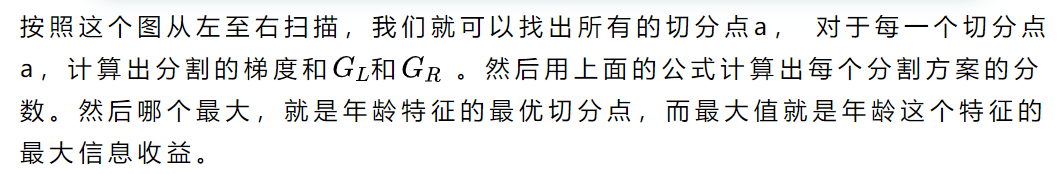


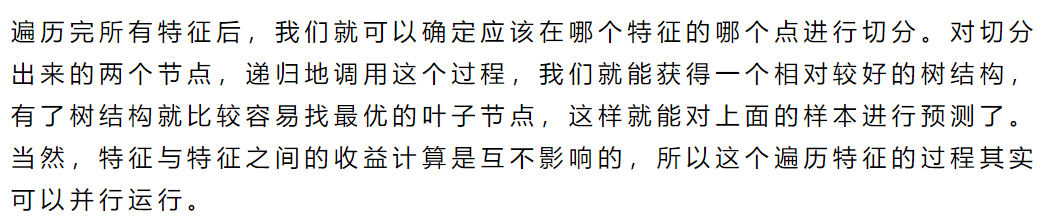
1. 那么我们就可以来梳理一下最优切分点的划分算法了：
   * 从深度为 0 的树开始，对每个叶节点枚举所有的可用特征；
   * 针对每个特征，把属于该节点的训练样本根据该特征值进行升序排列，通过线性扫描的方式来决定该特征的最佳分裂点，并记录该特征的分裂收益；（这个过程每个特征的收益计算是可以并行计算的，xgboost之所以快，其中一个原因就是因为它支持并行计算，而这里的并行正是指的特征之间的并行计算，千万不要理解成各个模型之间的并行）
   * 选择收益最大的特征作为分裂特征，用该特征的最佳分裂点作为分裂位置，在该节点上分裂出左右两个新的叶节点，并为每个新节点关联对应的样本集（这里稍微提一下，xgboost是可以处理空值的，也就是假如某个样本在这个最优分裂点上值为空的时候， 那么xgboost先把它放到左子树上计算一下收益，再放到右子树上计算收益，哪个大就把它放到哪棵树上。）
   * 回到第 1 步，递归执行到满足特定条件为止

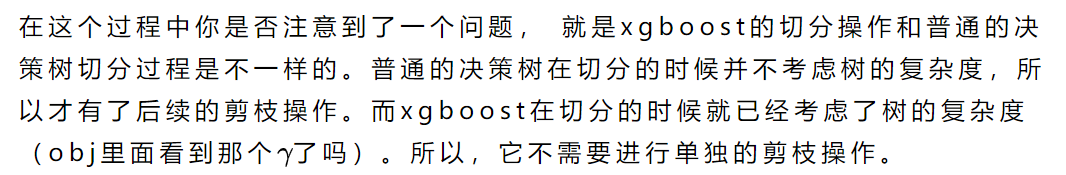
上面就是最优切分点划分算法的过程， 看完之后，是不是依然懵逼， 这到底是怎么做的啊， 下面就看一个寻找最优切分点的栗子吧：  
还是上面玩游戏的那个例子，假设我有这一家子人样本，每个人有性别，年龄，兴趣等几个特征，我想用xgboost建立一棵树预测玩游戏的意愿值。首先，五个人都聚集在根节点上，现在就考虑根节点分叉，我们就遍历每个特征，对于当前的特征，我们要去寻找最优切分点以及带来的最大收益，比如当前特征是年龄，我们需要知道两点：\* 按照年龄分是否有效，也就是是否减少了obj的值 \* 如果真的可以分，特征收益比较大， 那么我们从哪个年龄点分开呢？

对于这两个问题，我们可以这样做，首先我们先把年龄进行一个排序， 如下图：

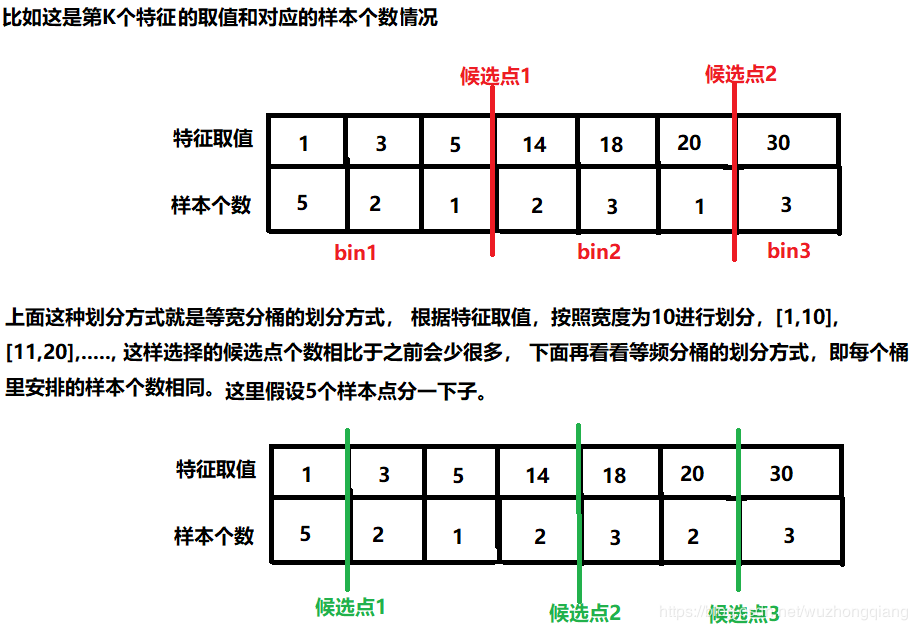


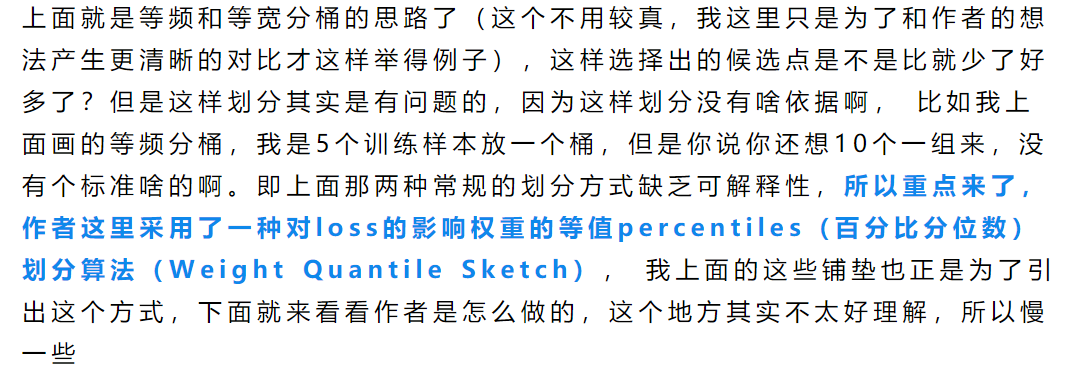


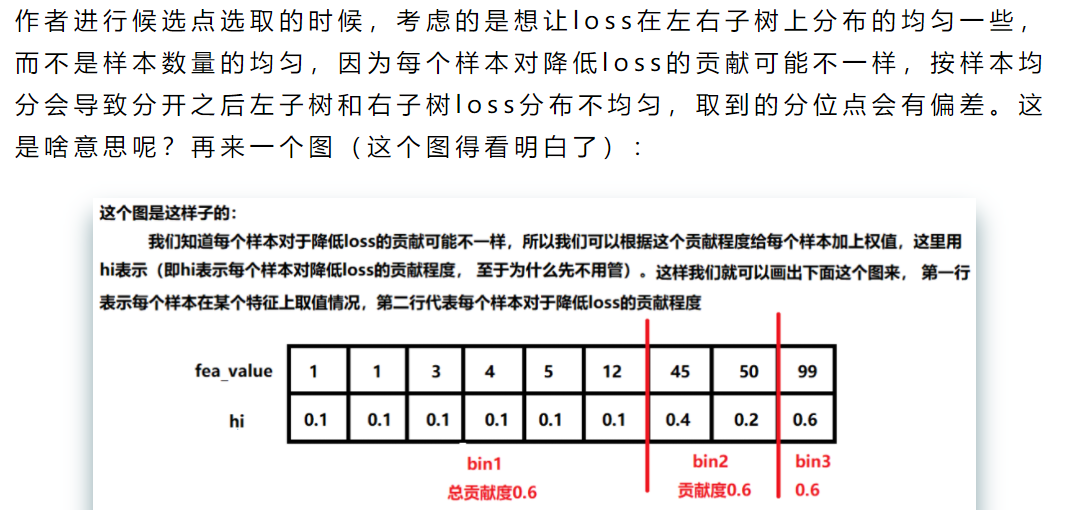




这就是xgboost贪心建树的一个思路了，即**遍历所有特征以及所有分割点，每次选最好的那个**。GBDT也是采用的这种方式， 这算法的确不错，但是有个问题你发现了没？就是计算代价太大了，尤其是数据量很大，分割点很多的时候，计算起来非常复杂并且也无法读入内存进行计算。所以作者想到了一种近似分割的方式（可以理解为分割点分桶的思路），选出一些候选的分裂点，然后再遍历这些较少的分裂点来找到最佳分裂点。那么怎么进行分桶选候选分裂点才比较合理呢？我们一般的思路可能是根据特征值的大小直接进行等宽或者等频分桶， 像下面这样（这个地方理解起来有点难，得画画了，图可能不太好看，能说明问题就行，哈哈）：







下面看看xgboost相比于GBDT有哪些优点（面试的时候可能会涉及）：

* 精度更高：GBDT只用到一阶泰勒， 而xgboost对损失函数进行了二阶泰勒展开， 一方面为了增加精度， 另一方面也为了能够自定义损失函数，二阶泰勒展开可以近似大量损失函数
* 灵活性更强：GBDT以CART作为基分类器，而Xgboost不仅支持CART，还支持线性分类器，另外，Xgboost支持自定义损失函数，只要损失函数有一二阶导数。
* 正则化：xgboost在目标函数中加入了正则，用于控制模型的复杂度。有助于降低模型方差，防止过拟合。正则项里包含了树的叶子节点个数，叶子节点权重的L2范式。
* Shrinkage（缩减）：相当于学习速率。这个主要是为了削弱每棵树的影响，让后面有更大的学习空间，学习过程更加的平缓
* 列抽样：这个就是在建树的时候，不用遍历所有的特征了，可以进行抽样，一方面简化了计算，另一方面也有助于降低过拟合
* 缺失值处理：这个是xgboost的稀疏感知算法，加快了节点分裂的速度
* 并行化操作：块结构可以很好的支持并行计算

# 2.20面试题

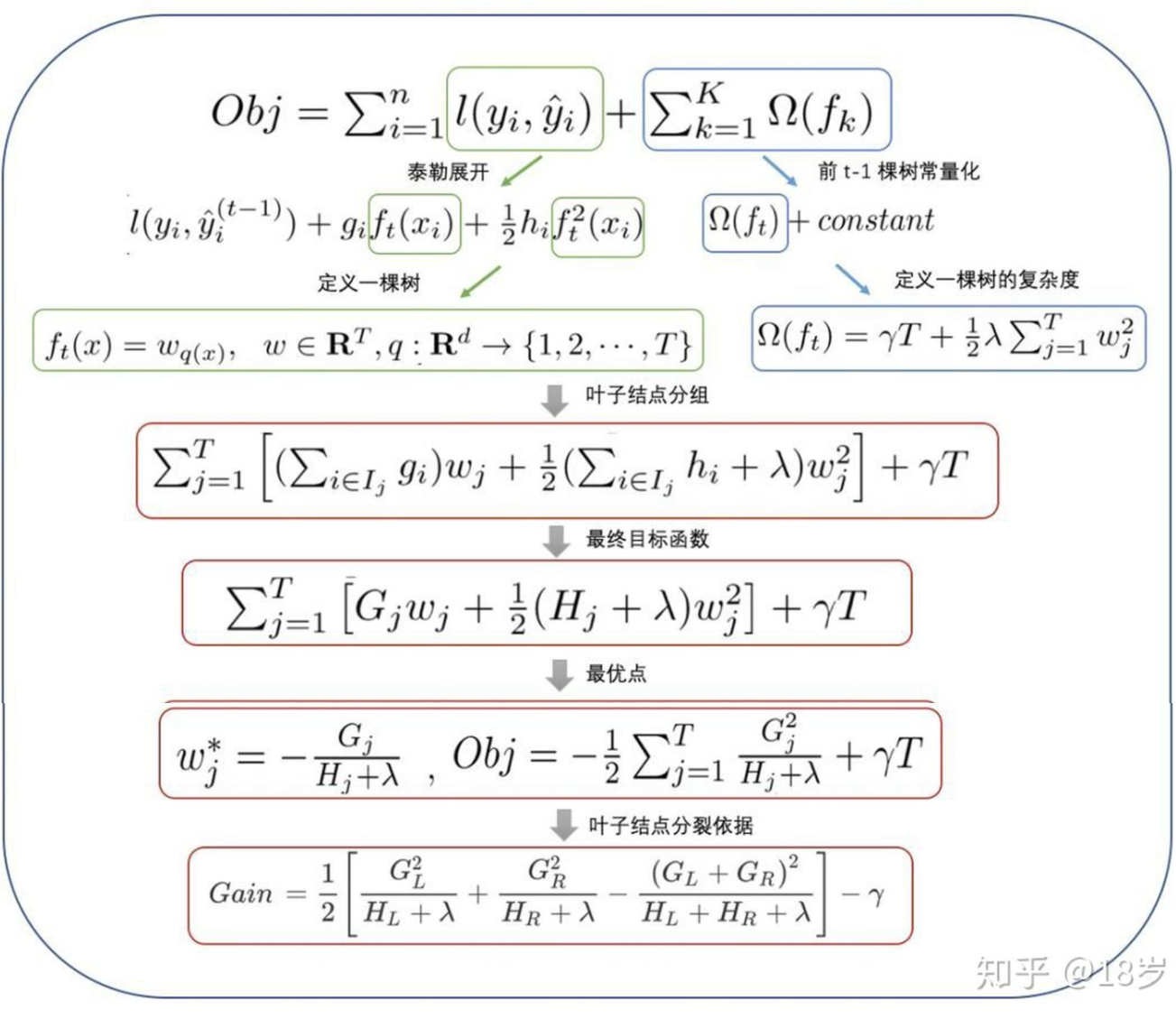
<https://mp.weixin.qq.com/s?__biz=MzI1MzY0MzE4Mg==&mid=2247485159&idx=1&sn=d429aac8370ca5127e1e786995d4e8ec&chksm=e9d01626dea79f30043ab80652c4a859760c1ebc0d602e58e13490bf525ad7608a9610495b3d&scene=21#wechat_redirect>

# 3.原理和示例

<https://zhuanlan.zhihu.com/p/92837676>

<https://www.jianshu.com/p/ac1c12f3fba1>

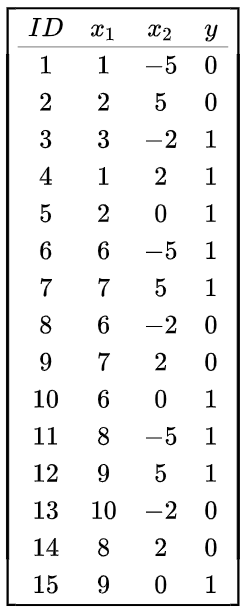
详细示例



**手动还原XGBoost实例过程**

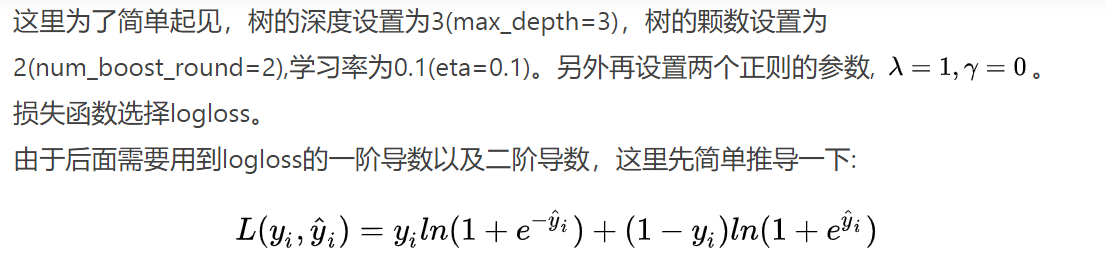
下面这个例子由<https://blog.csdn.net/qq_22238533/article/details/79477547>给出。

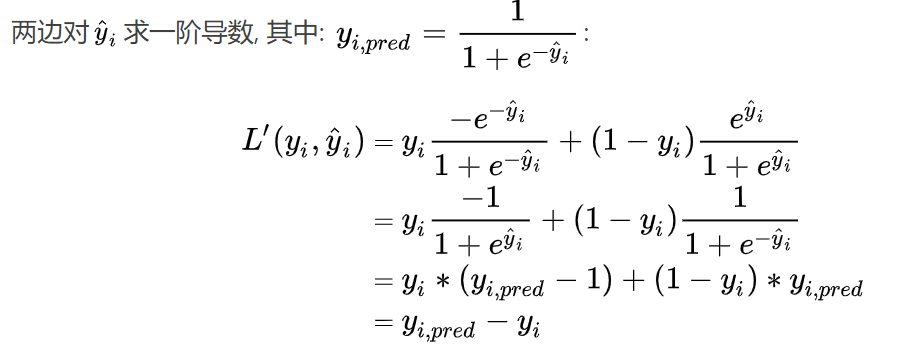
上面，我只是简单的阐述整个流程，有一些细节的地方可能都说的不太清楚，我以一个简单的UCI数据集，一步一步的演算整个xgboost的过程。数据集如下:

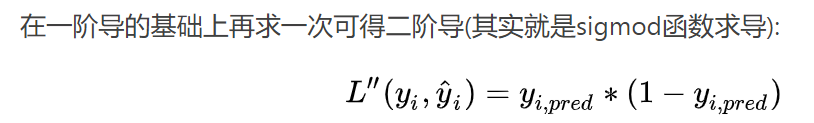
x1,x2是特征;

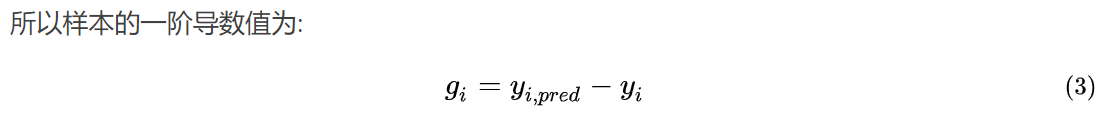
那么把样本如何分成两个集合呢？这里就是上面说到的选取一个最佳的特征以及分裂点使得Gain最大。

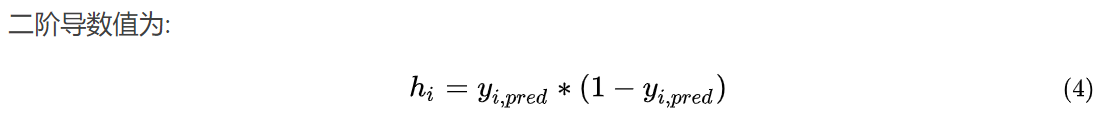
比如说对于特征x1，一共有[1,2,3,6,7,8,9,10]8种取值(注意上表的ID不代表取值)。可以得到以下这么多划分方式

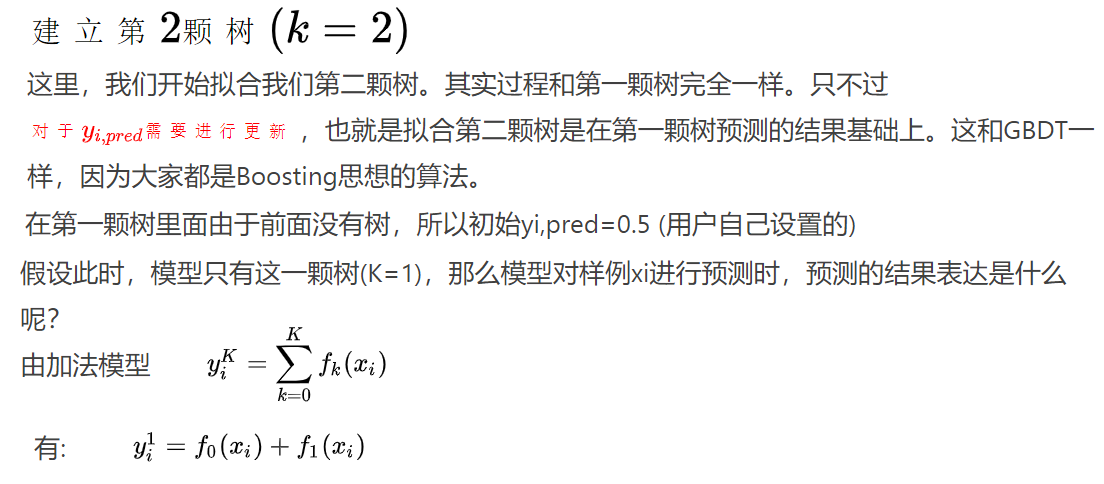


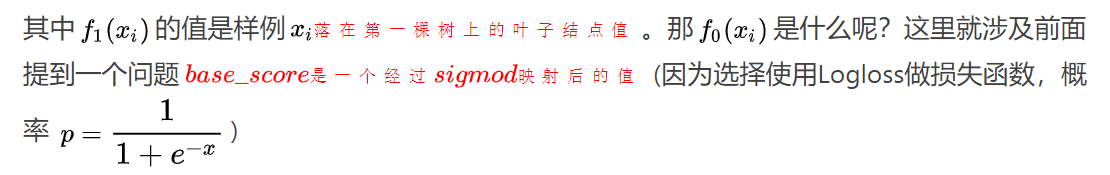


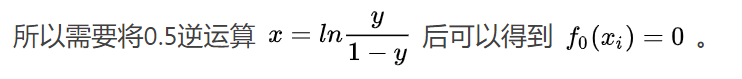


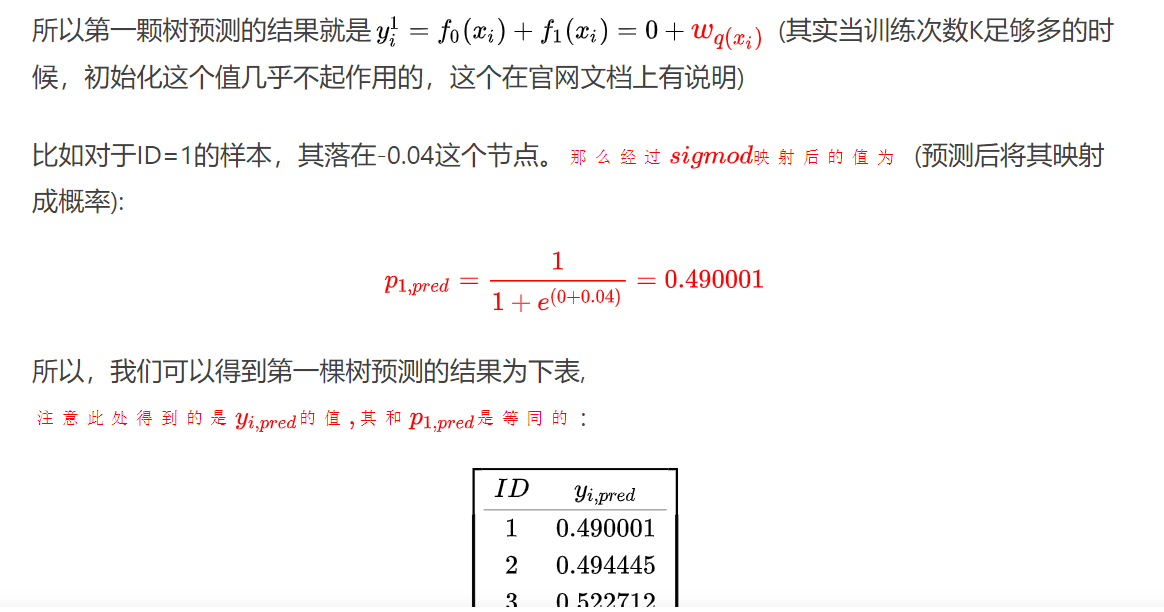


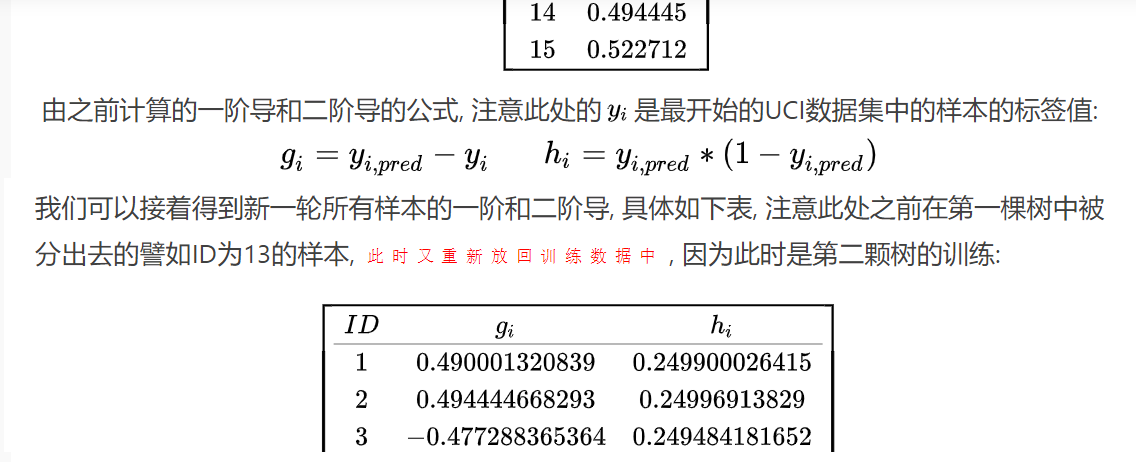












# xgboost与传统GBDT的区别与联系？

至此，我们来简单的总结一下xgboost和GBDT的区别以及联系。

区别：  
1.xgboost和GBDT的一个区别在于目标函数上。  
在xgboost中，损失函数+正则项。 GBDT中一般只有损失函数。  
2.xgboost中利用二阶导数的信息，而GBDT只利用了一阶导数, 即在GBDT回归中利用了残差的概念。  
3.xgboost在建树的时候利用的准则来源于目标函数推导，即可以理解为牛顿法。而GBDT建树利用的是启发式准则。  
4.xgboost中可以自动处理空缺值，自动学习空缺值的分裂方向，GBDT(sklearn版本)不允许包含空缺值。

相似点：  
1.xgboost和GBDT的学习过程都是一样的，都是基于Boosting的思想，先学习前n-1个学习器，然后基于前n-1个学习器学习第n个学习器。而且其都是将损失函数和分裂点评估函数分开了。  
2.建树过程都利用了损失函数的导数信息,。  
3.都使用了学习率来进行Shrinkage，从前面我们能看到不管是GBDT还是xgboost，我们都会利用学习率对拟合结果做缩减以减少过拟合的风险。

# Gbdt调参;demo;

可以resume训练??

param\_test1 = {'n\_estimators':range(20,81,10)}

gsearch1 = GridSearchCV(estimator = GradientBoostingClassifier(learning\_rate=0.1, min\_samples\_split=300,

min\_samples\_leaf=20,max\_depth=8,max\_features='sqrt', subsample=0.8,random\_state=2019),

param\_grid = param\_test1, scoring='roc\_auc',iid=False,cv=5)

gsearch1.fit(X, y)

print(gsearch1.cv\_results\_['mean\_test\_score'], gsearch1.best\_params\_, gsearch1.best\_score\_)

[0.81264331 0.8131308 0.81313477 0.81276756 0.81276597 0.81117966 0.81096291]

{'n\_estimators': 40} 0.813134765625

===============

找到了一个合适的迭代次数60，现在我们开始对决策树进行调参。

**1.2 选择max\_depth和min\_samples\_split**

首先我们对决策树最大深度max\_depth和内部节点再划分所需最小样本数min\_samples\_split进行网格搜索：

param\_test2 = {'max\_depth':range(3,14,2),

'min\_samples\_split':range(100,801,200)}

gsearch2 = GridSearchCV(estimator = GradientBoostingClassifier(learning\_rate=0.1, n\_estimators=60, min\_samples\_leaf=20,

max\_features='sqrt', subsample=0.8, random\_state=2019),

param\_grid = param\_test2, scoring='roc\_auc',iid=False, cv=5)

gsearch2.fit(X, y)

print(gsearch2.cv\_results\_['mean\_test\_score'], gsearch2.best\_params\_, gsearch2.best\_score\_)

输出：

{'max\_depth': 7, 'min\_samples\_split': 300}, 0.8213724275914632

可见最好的最大树深度是7，内部节点再划分所需最小样本数是300。

**五、特征重要度**

基于决策树的算法都可以输出特征的重要度，其中sklearn中GBDT的特征重要度函数为feature\_importances\_，lightgbm中的特征重要度函数为feature\_importance，这里以sklearn中的GBDT为例，输出特征重要度：

def plot\_feature\_importance(dataset, model\_bst):

list\_feature\_name = list(dataset.columns[:])

# list\_feature\_importance = list(model\_bst.feature\_importance(importance\_type='split', iteration=-1))

list\_feature\_importance = list(model\_bst.feature\_importances\_)

dataframe\_feature\_importance = pd.DataFrame(

{'feature\_name': list\_feature\_name, 'importance': list\_feature\_importance})

dataframe\_feature\_importance20 = dataframe\_feature\_importance.sort\_values(by='importance', ascending=False)[:20]

print(dataframe\_feature\_importance20)

x = range(len(dataframe\_feature\_importance20['feature\_name']))

plt.xticks(x, dataframe\_feature\_importance20['feature\_name'], rotation=90, fontsize=8)

plt.plot(x, dataframe\_feature\_importance20['importance'])

plt.xlabel("Feature name")

plt.ylabel("Importance")

plt.title("The importance of features")

plt.show()

gbm6 = GradientBoostingClassifier(learning\_rate=0.05, n\_estimators=160,max\_depth=7, min\_samples\_leaf =60,

min\_samples\_split =1200, max\_features=9, subsample=0.7, random\_state=2019)

gbm6.fit(x\_train, y\_train)

plot\_feature\_importance(x\_train, gbm6)

# sklearn demo

<https://scikit-learn.org/stable/auto_examples/ensemble/plot_gradient_boosting_regression.html>

**import** **matplotlib.pyplot** **as** **plt**

**import** **numpy** **as** **np**

**from** **sklearn** **import** datasets, ensemble

**from** **sklearn.inspection** **import** [permutation\_importance](https://scikit-learn.org/stable/modules/generated/sklearn.inspection.permutation_importance.html#sklearn.inspection.permutation_importance)

**from** **sklearn.metrics** **import** [mean\_squared\_error](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html#sklearn.metrics.mean_squared_error)

**from** **sklearn.model\_selection** **import** [train\_test\_split](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html#sklearn.model_selection.train_test_split)

diabetes = [datasets.load\_diabetes](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_diabetes.html#sklearn.datasets.load_diabetes)()

X, y = diabetes.data, diabetes.target

X\_train, X\_test, y\_train, y\_test = [train\_test\_split](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html#sklearn.model_selection.train_test_split)(

X, y, test\_size=0.1, random\_state=13)

params = {'n\_estimators': 500,

'max\_depth': 4,

'min\_samples\_split': 5,

'learning\_rate': 0.01,

'loss': 'ls'}

reg = [ensemble.GradientBoostingRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html#sklearn.ensemble.GradientBoostingRegressor)(\*\*params)

reg.fit(X\_train, y\_train)

mse = [mean\_squared\_error](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html#sklearn.metrics.mean_squared_error)(y\_test, reg.predict(X\_test))

print("The mean squared error (MSE) on test set: *{:.4f}*".format(mse))

test\_score = [np.zeros](https://numpy.org/doc/stable/reference/generated/numpy.zeros.html#numpy.zeros)((params['n\_estimators'],), dtype=[np.float64](https://numpy.org/doc/stable/reference/arrays.scalars.html#numpy.float64))

**for** i, y\_pred **in** enumerate(reg.staged\_predict(X\_test)):

test\_score[i] = reg.loss\_(y\_test, y\_pred)

fig = [plt.figure](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.figure.html#matplotlib.pyplot.figure)(figsize=(6, 6))

[plt.subplot](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html#matplotlib.pyplot.subplot)(1, 1, 1)

[plt.title](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html#matplotlib.pyplot.title)('Deviance')

[plt.plot](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)([np.arange](https://numpy.org/doc/stable/reference/generated/numpy.arange.html#numpy.arange)(params['n\_estimators']) + 1, reg.train\_score\_, 'b-',

label='Training Set Deviance')

[plt.plot](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)([np.arange](https://numpy.org/doc/stable/reference/generated/numpy.arange.html#numpy.arange)(params['n\_estimators']) + 1, test\_score, 'r-',

label='Test Set Deviance')

[plt.legend](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.legend.html#matplotlib.pyplot.legend)(loc='upper right')

[plt.xlabel](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlabel.html#matplotlib.pyplot.xlabel)('Boosting Iterations')

[plt.ylabel](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.ylabel.html#matplotlib.pyplot.ylabel)('Deviance')

fig.tight\_layout()

[plt.show](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.show.html#matplotlib.pyplot.show)()

feature\_importance = reg.feature\_importances\_

sorted\_idx = [np.argsort](https://numpy.org/doc/stable/reference/generated/numpy.argsort.html#numpy.argsort)(feature\_importance)

pos = [np.arange](https://numpy.org/doc/stable/reference/generated/numpy.arange.html#numpy.arange)(sorted\_idx.shape[0]) + .5

fig = [plt.figure](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.figure.html#matplotlib.pyplot.figure)(figsize=(12, 6))

[plt.subplot](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html#matplotlib.pyplot.subplot)(1, 2, 1)

[plt.barh](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.barh.html#matplotlib.pyplot.barh)(pos, feature\_importance[sorted\_idx], align='center')

[plt.yticks](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.yticks.html#matplotlib.pyplot.yticks)(pos, [np.array](https://numpy.org/doc/stable/reference/generated/numpy.array.html#numpy.array)(diabetes.feature\_names)[sorted\_idx])

[plt.title](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html#matplotlib.pyplot.title)('Feature Importance (MDI)')

result = [permutation\_importance](https://scikit-learn.org/stable/modules/generated/sklearn.inspection.permutation_importance.html#sklearn.inspection.permutation_importance)(reg, X\_test, y\_test, n\_repeats=10,

random\_state=42, n\_jobs=2)

sorted\_idx = result.importances\_mean.argsort()

[plt.subplot](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html#matplotlib.pyplot.subplot)(1, 2, 2)

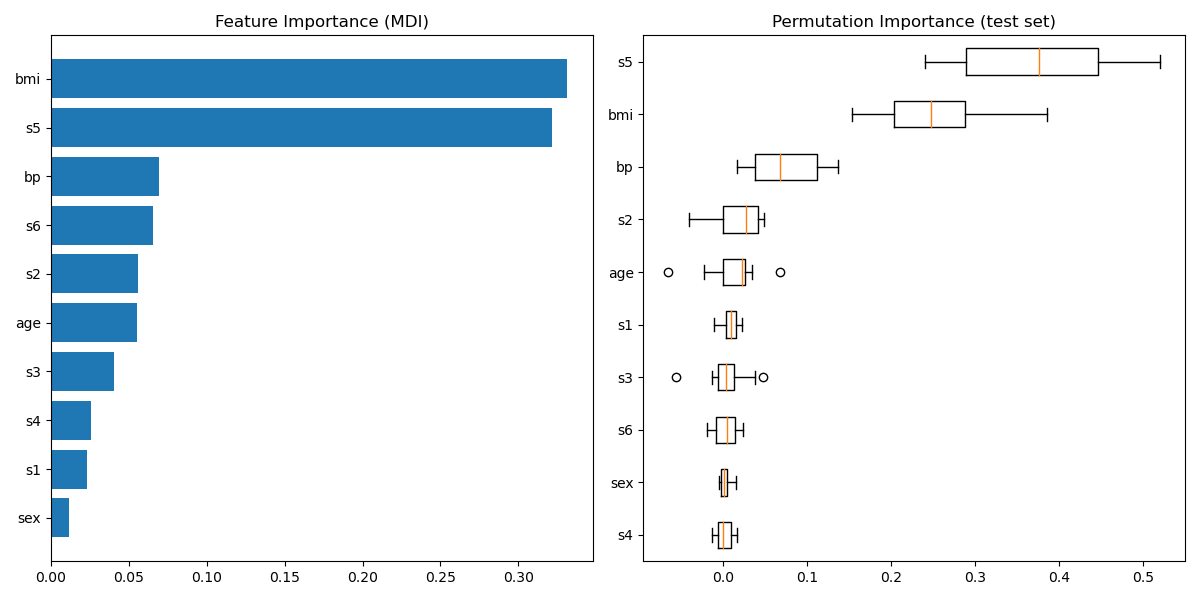
[plt.boxplot](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.boxplot.html#matplotlib.pyplot.boxplot)(result.importances[sorted\_idx].T,

vert=**False**, labels=[np.array](https://numpy.org/doc/stable/reference/generated/numpy.array.html" \l "numpy.array" \o "numpy.array)(diabetes.feature\_names)[sorted\_idx])

[plt.title](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html#matplotlib.pyplot.title)("Permutation Importance (test set)")

fig.tight\_layout()

[plt.show](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.show.html#matplotlib.pyplot.show)()



# Gbdt+lr; sklearn demo

Feature transformations with ensembles of trees

Transform your features into a higher dimensional, sparse space. Then train a linear model on these features.

<https://scikit-learn.org/stable/auto_examples/ensemble/plot_feature_transformation.html#example-ensemble-plot-feature-transformation-py>

First fit an ensemble of trees (totally random trees, a random forest, or gradient boosted trees) on the training set. Then each leaf of each tree in the ensemble is assigned a fixed arbitrary feature index in a new feature space. These leaf indices are then encoded in a one-hot fashion.

Each sample goes through the decisions of each tree of the ensemble and ends up in one leaf per tree. The sample is encoded by setting feature values for these leaves to 1 and the other feature values to 0.

The resulting transformer has then learned a supervised, sparse, high-dimensional categorical embedding of the data.

**import** **numpy** **as** **np**

[np.random.seed](https://numpy.org/doc/stable/reference/random/generated/numpy.random.seed.html#numpy.random.seed)(10)

**import** **matplotlib.pyplot** **as** **plt**

**from** **sklearn.datasets** **import** [make\_classification](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html#sklearn.datasets.make_classification)

**from** **sklearn.linear\_model** **import** [LogisticRegression](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression)

**from** **sklearn.ensemble** **import** ([RandomTreesEmbedding](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomTreesEmbedding.html" \l "sklearn.ensemble.RandomTreesEmbedding" \o "sklearn.ensemble.RandomTreesEmbedding), [RandomForestClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier),

[GradientBoostingClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html#sklearn.ensemble.GradientBoostingClassifier))

**from** **sklearn.preprocessing** **import** [OneHotEncoder](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html#sklearn.preprocessing.OneHotEncoder)

**from** **sklearn.model\_selection** **import** [train\_test\_split](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html#sklearn.model_selection.train_test_split)

**from** **sklearn.metrics** **import** [roc\_curve](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve)

**from** **sklearn.pipeline** **import** [make\_pipeline](https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.make_pipeline.html#sklearn.pipeline.make_pipeline)

n\_estimator = 10

X, y = [make\_classification](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html#sklearn.datasets.make_classification)(n\_samples=80000)

X\_train, X\_test, y\_train, y\_test = [train\_test\_split](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html#sklearn.model_selection.train_test_split)(X, y, test\_size=0.5)

*# It is important to train the ensemble of trees on a different subset*

*# of the training data than the linear regression model to avoid*

*# overfitting, in particular if the total number of leaves is*

*# similar to the number of training samples*

X\_train, X\_train\_lr, y\_train, y\_train\_lr = [train\_test\_split](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html#sklearn.model_selection.train_test_split)(

X\_train, y\_train, test\_size=0.5)

*# Unsupervised transformation based on totally random trees*

rt = [RandomTreesEmbedding](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomTreesEmbedding.html#sklearn.ensemble.RandomTreesEmbedding)(max\_depth=3, n\_estimators=n\_estimator,

random\_state=0)

rt\_lm = [LogisticRegression](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression)(max\_iter=1000)

pipeline = [make\_pipeline](https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.make_pipeline.html#sklearn.pipeline.make_pipeline)(rt, rt\_lm)

pipeline.fit(X\_train, y\_train)

y\_pred\_rt = pipeline.predict\_proba(X\_test)[:, 1]

fpr\_rt\_lm, tpr\_rt\_lm, \_ = [roc\_curve](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve)(y\_test, y\_pred\_rt)

*# Supervised transformation based on random forests*

rf = [RandomForestClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier)(max\_depth=3, n\_estimators=n\_estimator)

rf\_enc = [OneHotEncoder](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html#sklearn.preprocessing.OneHotEncoder)()

rf\_lm = [LogisticRegression](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression)(max\_iter=1000)

rf.fit(X\_train, y\_train)

rf\_enc.fit(rf.apply(X\_train))

rf\_lm.fit(rf\_enc.transform(rf.apply(X\_train\_lr)), y\_train\_lr)

y\_pred\_rf\_lm = rf\_lm.predict\_proba(rf\_enc.transform(rf.apply(X\_test)))[:, 1]

fpr\_rf\_lm, tpr\_rf\_lm, \_ = [roc\_curve](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve)(y\_test, y\_pred\_rf\_lm)

*# Supervised transformation based on gradient boosted trees*

grd = [GradientBoostingClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html#sklearn.ensemble.GradientBoostingClassifier)(n\_estimators=n\_estimator)

grd\_enc = [OneHotEncoder](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html#sklearn.preprocessing.OneHotEncoder)()

grd\_lm = [LogisticRegression](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression)(max\_iter=1000)

grd.fit(X\_train, y\_train)

grd\_enc.fit(grd.apply(X\_train)[:, :, 0])

grd\_lm.fit(grd\_enc.transform(grd.apply(X\_train\_lr)[:, :, 0]), y\_train\_lr)

y\_pred\_grd\_lm = grd\_lm.predict\_proba(

grd\_enc.transform(grd.apply(X\_test)[:, :, 0]))[:, 1]

fpr\_grd\_lm, tpr\_grd\_lm, \_ = [roc\_curve](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve)(y\_test, y\_pred\_grd\_lm)

*# The gradient boosted model by itself*

y\_pred\_grd = grd.predict\_proba(X\_test)[:, 1]

fpr\_grd, tpr\_grd, \_ = [roc\_curve](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve)(y\_test, y\_pred\_grd)

*# The random forest model by itself*

y\_pred\_rf = rf.predict\_proba(X\_test)[:, 1]

fpr\_rf, tpr\_rf, \_ = [roc\_curve](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve)(y\_test, y\_pred\_rf)

[plt.figure](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.figure.html#matplotlib.pyplot.figure)(1)

[plt.plot](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)([0, 1], [0, 1], 'k--')

[plt.plot](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(fpr\_rt\_lm, tpr\_rt\_lm, label='RT + LR')

[plt.plot](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(fpr\_rf, tpr\_rf, label='RF')

[plt.plot](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(fpr\_rf\_lm, tpr\_rf\_lm, label='RF + LR')

[plt.plot](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(fpr\_grd, tpr\_grd, label='GBT')

[plt.plot](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(fpr\_grd\_lm, tpr\_grd\_lm, label='GBT + LR')

[plt.xlabel](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlabel.html#matplotlib.pyplot.xlabel)('False positive rate')

[plt.ylabel](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.ylabel.html#matplotlib.pyplot.ylabel)('True positive rate')

[plt.title](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html#matplotlib.pyplot.title)('ROC curve')

[plt.legend](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.legend.html#matplotlib.pyplot.legend)(loc='best')

[plt.show](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.show.html#matplotlib.pyplot.show)()

[plt.figure](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.figure.html#matplotlib.pyplot.figure)(2)

[plt.xlim](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlim.html#matplotlib.pyplot.xlim)(0, 0.2)

[plt.ylim](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.ylim.html#matplotlib.pyplot.ylim)(0.8, 1)

[plt.plot](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)([0, 1], [0, 1], 'k--')

[plt.plot](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(fpr\_rt\_lm, tpr\_rt\_lm, label='RT + LR')

[plt.plot](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(fpr\_rf, tpr\_rf, label='RF')

[plt.plot](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(fpr\_rf\_lm, tpr\_rf\_lm, label='RF + LR')

[plt.plot](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(fpr\_grd, tpr\_grd, label='GBT')

[plt.plot](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)(fpr\_grd\_lm, tpr\_grd\_lm, label='GBT + LR')

[plt.xlabel](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlabel.html#matplotlib.pyplot.xlabel)('False positive rate')

[plt.ylabel](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.ylabel.html#matplotlib.pyplot.ylabel)('True positive rate')

[plt.title](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html#matplotlib.pyplot.title)('ROC curve (zoomed in at top left)')

[plt.legend](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.legend.html#matplotlib.pyplot.legend)(loc='best')

[plt.show](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.show.html#matplotlib.pyplot.show)()

