我的XGBoost学习经历及动手实践

原创 李祖贤 Datawhale 2020-06-21

收录于话题
#机器学习专题 19 #数据项目专栏 13

↑↑↑关注后"星标"Datawhale
每日干货 & 每月组队学习，不错过

Datawhale干货

作者：李祖贤 深圳大学，Datawhale高校群成员

知乎地址：http://www.zhihu.com/people/meng-di-76-92

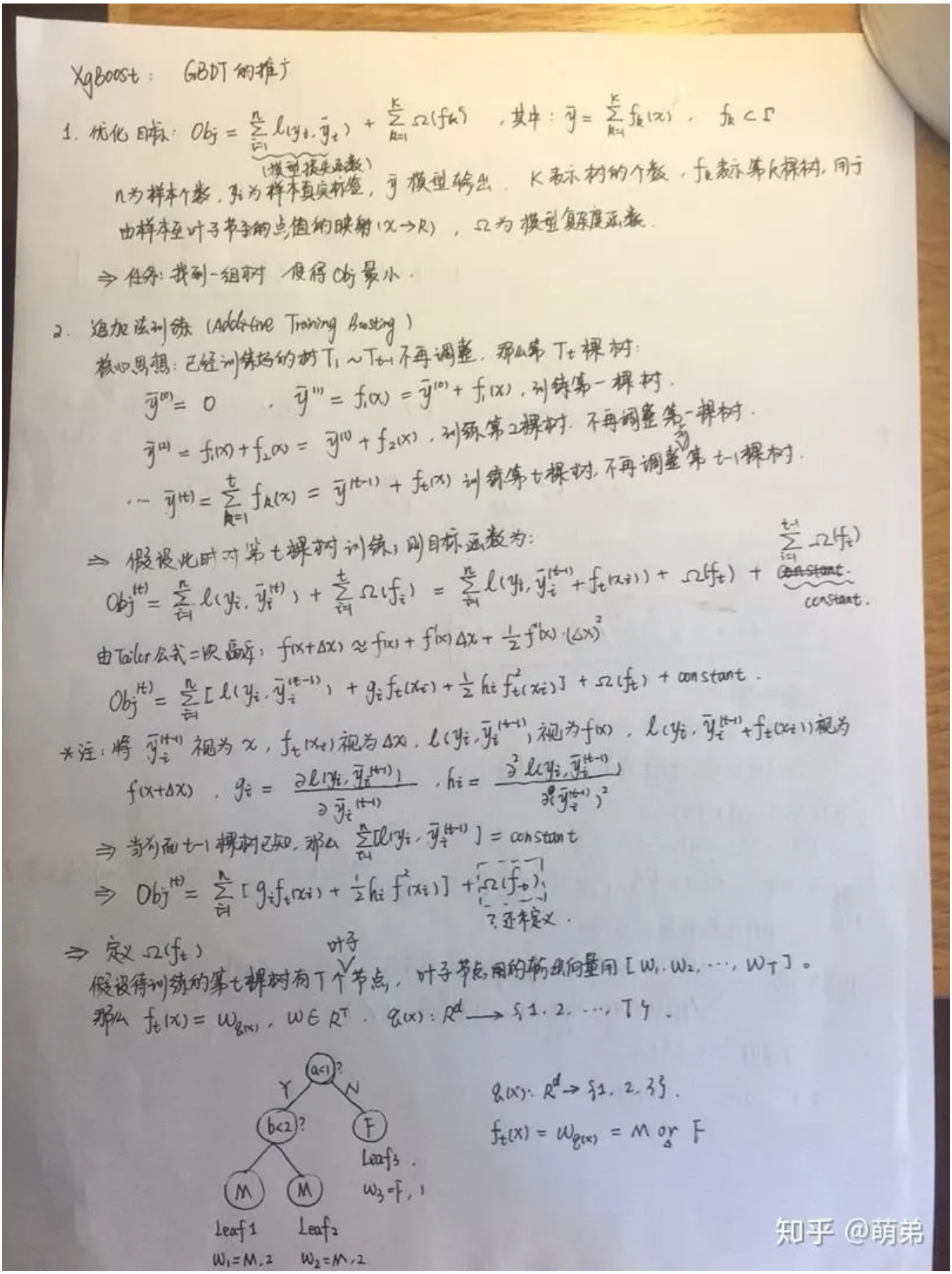
我今天主要介绍机器学习集成学习方法中三巨头之一的XGBoost，这个算法在早些时候机器学习比赛内曾经大放异彩，是非常好用的一个机器学习集成算法。

XGBoost是一个优化的分布式梯度增强库，旨在实现高效，灵活和便携。它在Gradient Boosting框架下实现机器学习算法。XGBoost提供了并行树提升（也称为GBDT，GBM），可以快速准确地解决许多数据科学问题。

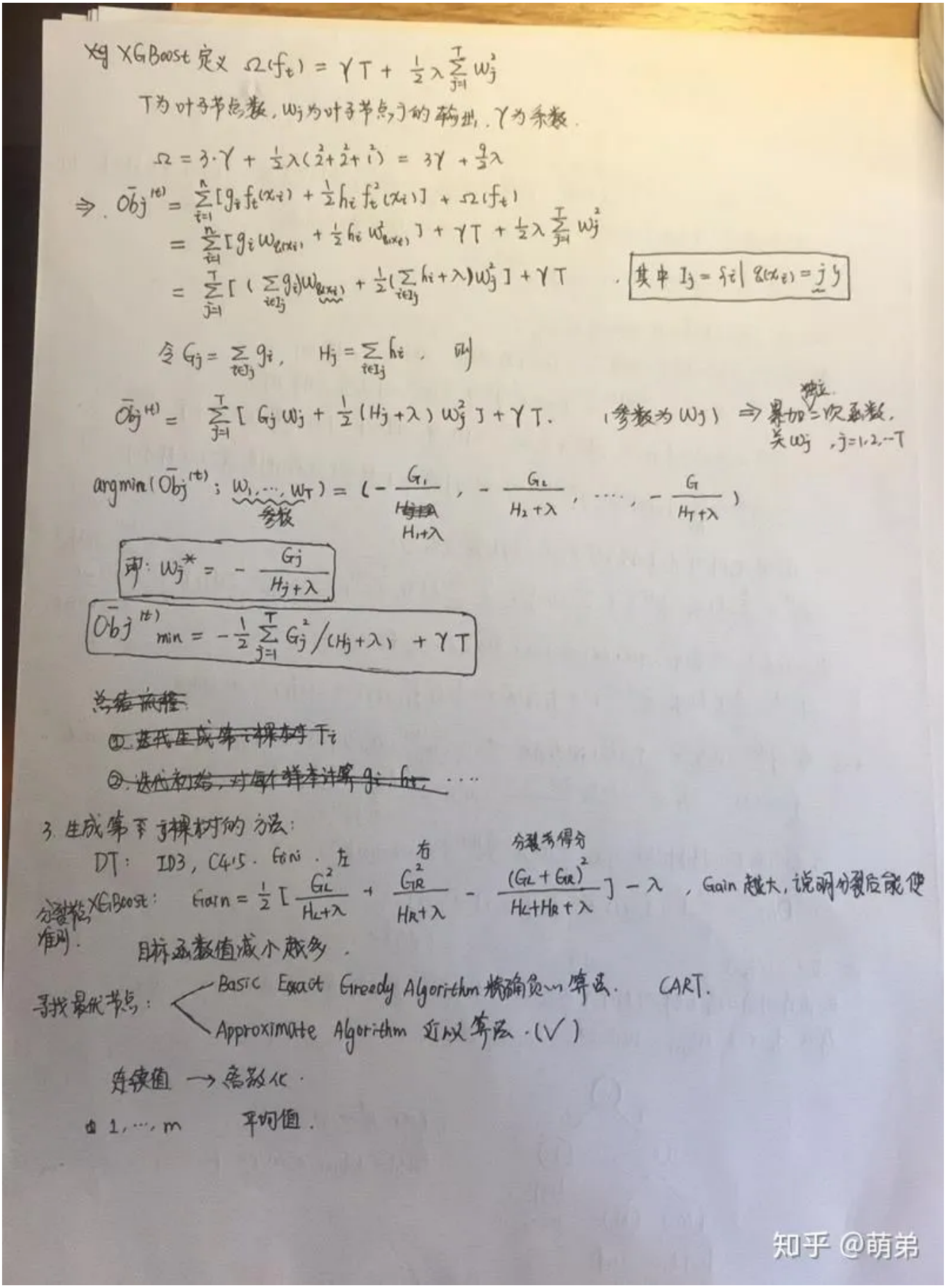
相同的代码在主要的分布式环境（Hadoop，SGE，MPI）上运行，并且可以解决超过数十亿个样例的问题。XGBoost利用了核外计算并且能够使数据科学家在一个主机上处理数亿的样本数据。最终，将这些技术进行结合来做一个端到端的系统以最少的集群系统来扩展到更大的数据集上。

XGBoost原理介绍

从0开始学习，经历过推导公式的波澜曲折，下面展示下我自己的推公式的手稿吧，希望能激励到大家能够对机器学习数据挖掘更加热爱！



XGBoost公式1



XGBoost公式2

现在我们对手稿的内容进行详细的讲解:

1. 优化目标:

$$Obj = \sum_{i=1}^{n} l(y_i, \bar{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

其中，n为样本个数，y_i为第i个样本真实标签，$\bar{y}_i$表示模型的第i个样本输出值，K表示树的个数，f_k表示第k棵树，用于样本到叶子结点的映射$(x \to R), \Omega$为模型复杂度函数。

我们的任务是找到一组树使得OBj最小，很明显这个优化目标OBj可以看成是样本的损失和模型的复杂度惩罚相加组成。

2. 使用追加法训练（Additive Training Boosting）

核心思想是：在已经训练好了 $T_1 \sim T_{t-1}$ 棵树后不再调整前 T_{t-1} 棵树，那么第t棵树可以表示为：

$$\bar{y}^{(t)} = \sum_{k=1}^{t} f_k(x) = \bar{y}^{(t-1)} + f_t(x)$$

(1). 那此时如果我们对第t棵树训练，则目标函数为：

$$Obj^{(t)} = \sum_{i=1}^{n} l(y_i, \bar{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) = \sum_{i=1}^{n} l(y_i, \bar{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + \boxed{\sum_{i=1}^{t-1} \Omega(f_i) \quad 前t-1棵树已知下为常数}$$

对上式进行泰勒二阶展开：

$$Obj^{(t)} = \sum_{i=1}^{n} [l(y_i, \bar{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + constant$$

$*$注：$Tailor$公式是$f(x + \triangle x) \approx f(x) + f'(x)\triangle x + \frac{1}{2} f''(x)\triangle x^2$,

将$\bar{y}_i^{(t-1)}$视为x，$f_t(x_i)$视为$\triangle x, l(y_i, \bar{y}_i^{(t-1)})$视为$f(x)$，$l(y_i, \bar{y}_i^{(t-1)} + f_t(x_i))$视为$f(x + \triangle x)$,

$$其中 g_i = \frac{\partial\, l(y_i, \bar{y}_i^{(t-1)})}{\partial\, \bar{y}_i^{(t-1)}}, h_i = \frac{\partial^2\, l(y_i, \bar{y}_i^{(t-1)})}{\partial^2\, \bar{y}_i^{(t-1)}}$$

由于前t-1棵树已知，那么

$\sum_{i=1}^{n} l(y_i, \bar{y}_i^{(t-1)}) = constant$常数，常数不影响我们的优化故删除，得：

$$Obj^{(t)} = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

(2). 我们已经对前半部分的损失函数做出了充分的讨论，但是后半部分的 $\Omega(f_t)$ 还只是个符号并未定义，那我们现在就来定义 $\Omega(f_t)$ ：假设我们待训练的第t棵树有T个叶子结点:叶子结点的输出向量表示如下：

$$[w_1, w_2, \ldots, w_T]$$

假设 $q(x) : R^d \to \{1, 2, 3, \ldots, T\}$ 表示样本到叶子结点的映射，那么 $f_t(x) = w_{q(x)}, w \in R^T$ 。

那么我们定义 ：

$\Omega(f_t)=\gamma T+\frac{1}{2}\lambda\sum_{j=1}^{T}w_j^2$,其中$T$为叶子结点数，$w_j$为叶子结点$j$的输出，$\gamma$为系数

(3). 我们的目标函数最终化简为：

$$OBj^{(t)}=\sum_{i=1}^{n}[g_i f_t(x_i)+\frac{1}{2}h_i f_t^2(x_i)]+\gamma T+\frac{1}{2}\lambda\sum_{j=1}^{T}w_j^2$$

$$=\sum_{j=1}^{T}[(\sum_{i\in I_j}g_i)w_{q_{(x_i)}}+\frac{1}{2}(\sum_{i\in I_j}h_i+\lambda)w_j^2]+\gamma T$$

其中，$I_j=\{i|q(x_i)=j\}$,令$G_j=\sum_{i\in I_j}g_i$，$H_j=\sum_{i\in I_j}h_i$，则最终

$$\boxed{OBj^{(t)}=\sum_{j=1}^{T}[G_j w_j+\frac{1}{2}(H_j+\lambda)w_j^2]+\gamma T}$$

我们找到了目标函数就需要对目标函数进行优化:

$$argmin(OBj^{(t)};w_1,\ldots,w_T)=argmin(\sum_{j=1}^{T}[G_j w_j+\frac{1}{2}(H_j+\lambda)w_j^2]+\gamma T)$$

我们可以看到，这是一个累加独立的二次函数，因此根据二次函数最小点$x=-\frac{b}{2a}$得：

$$\boxed{\text{最优的参数}w_j^*=-\frac{G_j}{H_j+\lambda}\text{，最小值为：}OBj_{min}^{(t)}=-\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda}+\gamma T}$$

3. 生成树的策略:

我们刚刚的假设前提是已知前t-1棵树，因此我们现在来探讨怎么生成树。根据决策树的生成策略，再每次分裂节点的时候我们需要考虑能使得损失函数减小最快的节点，也就是分裂后损失函数减去分裂前损失函数我们称之为Gain：

$$\text{某个节点的}\boxed{Gain=\frac{1}{2}[\frac{G_L^2}{H_L+\lambda}+\frac{G_R^2}{H_R+\lambda}-\frac{(G_L+G_R)^2}{H_L+H_R+\lambda}]-\lambda}$$

Gain越大越能说明分裂后目标函数值减小越多。（因为从式子来看：$\frac{G_j^2}{H_j+\lambda}$ 越大，反而OBj越小）

4. 寻找最优节点:

- 精确贪心算法（Basic Exact Greedy Algorithm）

- 近似算法（Approximate Algorithm）

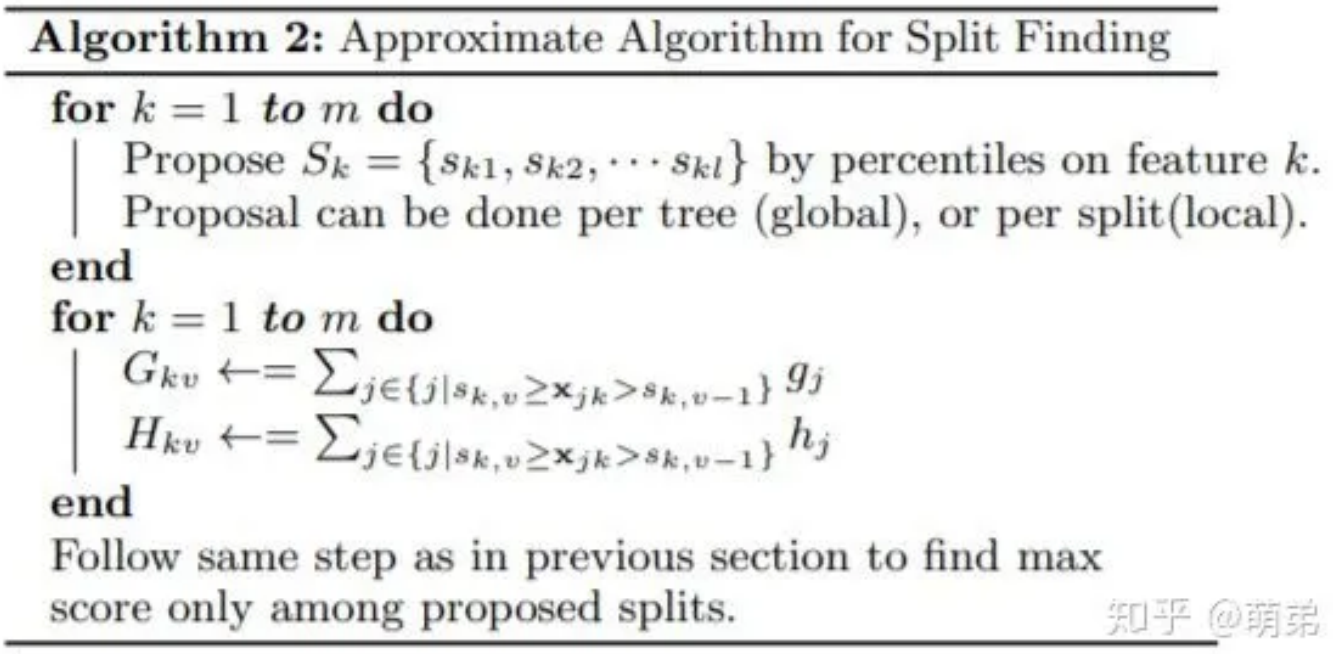
在决策树（CART）里面，我们使用的是精确贪心算法（Basic Exact Greedy Algorithm）,也就是将所有特征的所有取值排序（耗时耗内存巨大），然后比较每一个点的Gini，找出变化最大的节点。当特征是连续特征时，我们对连续值离散化，取两点的平均值为分割节点。可以看到，这里的排序算法需要花费大量的时间，因为要遍历整个样本所有特征，而且还要排序！！

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ ***to*** m **do**
  $G_L \leftarrow 0$, $H_L \leftarrow 0$
  for j *in sorted*$(I,$ *by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

论文的精确贪心算法的伪代码

因此在XGBoost里面我们使用的是近似算法（Approximate Algorithm）：该算法首先根据特征分布的百分位数(percentiles)提出候选分裂点，将连续特征映射到由这些候选点分割的桶中，汇总统计信息并根据汇总的信息在提案中找到最佳解决方案。对于某个特征k，算法首先根据特征分布的分位数找到特征切割点的候选集合 $S_k = \{S_{k_1}, S_{k_2}, \ldots, S_{k_l}\}$,然后将特征k的值根据集合 S_k 划分到桶(bucket)中，接着对每个桶内的样本统计值G、H进行累加，最后在这些累计的统计量上寻找最佳分裂点。



论文的近似算法的伪代码

XGBoost动手实践：

1. 引入基本工具库：

```
# 引入基本工具库
import numpy as np
import pandas as pd
import xgboost as xgb
import matplotlib.pyplot as plt
plt.style.use("ggplot")
%matplotlib inline
```

2. XGBoost原生工具库的上手：

```
import xgboost as xgb  # 引入工具库
# read in data
dtrain = xgb.DMatrix('demo/data/agaricus.txt.train')    # XGBoost的专属数据格式,
dtest = xgb.DMatrix('demo/data/agaricus.txt.test')  # # XGBoost的专属数据格式,
# specify parameters via map
param = {'max_depth':2, 'eta':1, 'objective':'binary:logistic' }    # 设置XGB
num_round = 2     # 使用线程数
bst = xgb.train(param, dtrain, num_round)    # 训练
# make prediction
preds = bst.predict(dtest)    # 预测
```

3. XGBoost的参数设置(括号内的名称为sklearn接口对应的参数名字)

XGBoost的参数分为三种：

1. 通用参数

- booster:使用哪个弱学习器训练，默认gbtree，可选gbtree，gblinear 或dart
- nthread：用于运行XGBoost的并行线程数，默认为最大可用线程数
- verbosity：打印消息的详细程度。有效值为0（静默），1（警告），2（信息），3（调试）。
- Tree Booster的参数：
  - eta（learning_rate）：learning_rate，在更新中使用步长收缩以防止过度拟合，

通用参数有两种类型的booster，因为tree的性能比线性回归好得多，因此我们很少用线性回归。

2. 任务参数

差，排名的平均平均精度），用户可以添加多个评估指标
  - rmse，均方根误差；rmsle：均方根对数误差；mae：平均绝对误差；mphe：平均伪Huber错误；logloss：负对数似然；error：二进制分类错误率；
  - merror：多类分类错误率；mlogloss：多类logloss；auc：曲线下面积；aucpr：PR曲线下的面积；ndcg：归一化累计折扣；map：平均精度；
- seed：随机数种子，[默认= 0]。

这个参数用来控制理想的优化目标和每一步结果的度量方法。

3. 命令行参数

这里不说了，因为很少用命令行控制台版本

4. XGBoost的调参说明：

参数调优的一般步骤：

- 1.确定（较大）学习速率和提升参数调优的初始值
- 2.max_depth 和 min_child_weight 参数调优
- 3.gamma参数调优
- 4.subsample 和 colsample_bytree 参数优
- 5.正则化参数alpha调优
- 6.降低学习速率和使用更多的决策树

5. XGBoost详细攻略：

1). 安装XGBoost

方式1：

```
pip3 install xgboost
```

方式2：

```
pip install xgboost
```

2). 数据接口（XGBoost可处理的数据格式DMatrix）

```
# 1.LibSVM文本格式文件
dtrain = xgb.DMatrix('train.svm.txt')
dtest = xgb.DMatrix('test.svm.buffer')
# 2.CSV文件(不能含类别文本变量，如果存在文本变量请做特征处理如one-hot)
dtrain = xgb.DMatrix('train.csv?format=csv&label_column=0')
dtest = xgb.DMatrix('test.csv?format=csv&label_column=0')
# 3.NumPy数组
data = np.random.rand(5, 10)  # 5 entities, each contains 10 features
label = np.random.randint(2, size=5)  # binary target
dtrain = xgb.DMatrix(data, label=label)
# 4.scipy.sparse数组
csr = scipy.sparse.csr_matrix((dat, (row, col)))
dtrain = xgb.DMatrix(csr)
# pandas数据框dataframe
data = pandas.DataFrame(np.arange(12).reshape((4,3)), columns=['a', 'b', 'c'])
label = pandas.DataFrame(np.random.randint(2, size=4))
dtrain = xgb.DMatrix(data, label=label)
```

笔者推荐：先保存到XGBoost二进制文件中将使加载速度更快，然后再加载进来

```
# 1.保存DMatrix到XGBoost二进制文件中
dtrain = xgb.DMatrix('train.svm.txt')
dtrain.save_binary('train.buffer')
# 2. 缺少的值可以用DMatrix构造函数中的默认值替换:
dtrain = xgb.DMatrix(data, label=label, missing=-999.0)
# 3.可以在需要时设置权重:
w = np.random.rand(5, 1)
```

```
dtrain = xgb.DMatrix(data, label=label, missing=-999.0, weight=w)
```

3). 参数的设置方式：

```
# 加载并处理数据
df_wine = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databas
df_wine.columns = ['Class label', 'Alcohol','Malic acid', 'Ash','Alcalinity of
                   'Flavanoids', 'Nonflavanoid phenols','Proanthocyanins','Col
df_wine = df_wine[df_wine['Class label'] != 1]  # drop 1 class
y = df_wine['Class label'].values
X = df_wine[['Alcohol','OD280/OD315 of diluted wines']].values
from sklearn.model_selection import train_test_split  # 切分训练集与测试集
from sklearn.preprocessing import LabelEncoder   # 标签化分类变量
le = LabelEncoder()
y = le.fit_transform(y)
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_stat
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test)
# 1.Booster 参数
params = {
    'booster': 'gbtree',
    'objective': 'multi:softmax',  # 多分类的问题
    'num_class': 10,               # 类别数，与 multisoftmax 并用
    'gamma': 0.1,                  # 用于控制是否后剪枝的参数,越大越保守，一般0.1、
    'max_depth': 12,               # 构建树的深度，越大越容易过拟合
    'lambda': 2,                   # 控制模型复杂度的权重值的L2正则化项参数，参数越
    'subsample': 0.7,              # 随机采样训练样本
    'colsample_bytree': 0.7,       # 生成树时进行的列采样
    'min_child_weight': 3,
    'silent': 1,                   # 设置成1则没有运行信息输出，最好是设置为0.
    'eta': 0.007,                  # 如同学习率
    'seed': 1000,
    'nthread': 4,                  # cpu 线程数
    'eval_metric':'auc'
}
plst = params.items()
# evallist = [(dtest, 'eval'), (dtrain, 'train')]   # 指定验证集
```

4). 训练

```
# 2.训练
num_round = 10
bst = xgb.train( plst, dtrain, num_round)
#bst = xgb.train( plst, dtrain, num_round, evallist )
```

5). 保存模型

```
# 3.保存模型
bst.save_model('0001.model')
# dump model
bst.dump_model('dump.raw.txt')
# dump model with feature map
#bst.dump_model('dump.raw.txt', 'featmap.txt')
```

6) . 加载保存的模型

```
# 4.加载保存的模型:
bst = xgb.Booster({'nthread': 4})  # init model
bst.load_model('0001.model')  # load data
```

7). 设置早停机制

```
# 5.也可以设置早停机制（需要设置验证集）
train(..., evals=evals, early_stopping_rounds=10)
```

8). 预测

```
# 6.预测
ypred = bst.predict(dtest)
```

9). 绘图

```
# 1.绘制重要性
xgb.plot_importance(bst)
# 2.绘制输出树
#xgb.plot_tree(bst, num_trees=2)
# 3.使用xgboost.to_graphviz()将目标树转换为graphviz
#xgb.to_graphviz(bst, num_trees=2)
```

6. 实战案例：

1). 分类案例

```
from sklearn.datasets import load_iris
import xgboost as xgb
from xgboost import plot_importance
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score   # 准确率
# 加载样本数据集
iris = load_iris()
X,y = iris.data,iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando

# 算法参数
params = {
    'booster': 'gbtree',
    'objective': 'multi:softmax',
    'num_class': 3,
    'gamma': 0.1,
    'max_depth': 6,
    'lambda': 2,
    'subsample': 0.7,
    'colsample_bytree': 0.75,
    'min_child_weight': 3,
    'silent': 0,
    'eta': 0.1,
    'seed': 1,
    'nthread': 4,
}

plst = params.items()

dtrain = xgb.DMatrix(X_train, y_train) # 生成数据集格式
num_rounds = 500
model = xgb.train(plst, dtrain, num_rounds) # xgboost模型训练

# 对测试集进行预测
dtest = xgb.DMatrix(X_test)
y_pred = model.predict(dtest)
```

```
38
39 # 计算准确率
40 accuracy = accuracy_score(y_test,y_pred)
41 print("accuarcy: %.2f%%" % (accuracy*100.0))
42
43 # 显示重要特征
44 plot_importance(model)
45 plt.show()
```

2). 回归案例

```
1  import xgboost as xgb
2  from xgboost import plot_importance
3  from matplotlib import pyplot as plt
4  from sklearn.model_selection import train_test_split
5  from sklearn.datasets import load_boston
6  from sklearn.metrics import mean_squared_error
7
8  # 加载数据集
9  boston = load_boston()
10 X,y = boston.data,boston.target
11
12 # XGBoost训练过程
13 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
14
15 params = {
16     'booster': 'gbtree',
17     'objective': 'reg:squarederror',
18     'gamma': 0.1,
19     'max_depth': 5,
20     'lambda': 3,
21     'subsample': 0.7,
22     'colsample_bytree': 0.7,
23     'min_child_weight': 3,
24     'silent': 1,
25     'eta': 0.1,
26     'seed': 1000,
27     'nthread': 4,
28 }
```

```
dtrain = xgb.DMatrix(X_train, y_train)
num_rounds = 300
plst = params.items()
model = xgb.train(plst, dtrain, num_rounds)

# 对测试集进行预测
dtest = xgb.DMatrix(X_test)
ans = model.predict(dtest)

# 显示重要特征
plot_importance(model)
plt.show()
```

7. XGBoost调参（结合sklearn网格搜索）

```
import xgboost as xgb
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score

iris = load_iris()
X,y = iris.data,iris.target
col = iris.target_names
train_x, valid_x, train_y, valid_y = train_test_split(X, y, test_size=0.3, ran
parameters = {
              'max_depth': [5, 10, 15, 20, 25],
              'learning_rate': [0.01, 0.02, 0.05, 0.1, 0.15],
              'n_estimators': [500, 1000, 2000, 3000, 5000],
              'min_child_weight': [0, 2, 5, 10, 20],
              'max_delta_step': [0, 0.2, 0.6, 1, 2],
              'subsample': [0.6, 0.7, 0.8, 0.85, 0.95],
              'colsample_bytree': [0.5, 0.6, 0.7, 0.8, 0.9],
              'reg_alpha': [0, 0.25, 0.5, 0.75, 1],
              'reg_lambda': [0.2, 0.4, 0.6, 0.8, 1],
              'scale_pos_weight': [0.2, 0.4, 0.6, 0.8, 1]

```

```
}

xlf = xgb.XGBClassifier(max_depth=10,
            learning_rate=0.01,
            n_estimators=2000,
            silent=True,
            objective='multi:softmax',
            num_class=3 ,
            nthread=-1,
            gamma=0,
            min_child_weight=1,
            max_delta_step=0,
            subsample=0.85,
            colsample_bytree=0.7,
            colsample_bylevel=1,
            reg_alpha=0,
            reg_lambda=1,
            scale_pos_weight=1,
            seed=0,
            missing=None)

gs = GridSearchCV(xlf, param_grid=parameters, scoring='accuracy', cv=3)
gs.fit(train_x, train_y)

print("Best score: %0.3f" % gs.best_score_)
print("Best parameters set: %s" % gs.best_params_ )
```