LightGBM:A Highly Efficient Gradient Boosting Decision Tree

王太峰 [徐静 整理]

2018/10/05

微软亚洲研究院AI头条分享

Introduction to LightGBM

Taifeng Wang (王太峰)

主管研究员

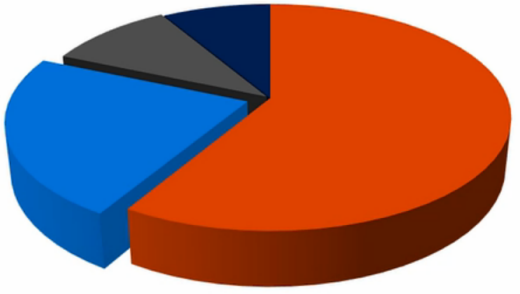
微软亚洲研究院机器学习组

Microsoft

DMTK http://dmtk.io

LightGBM

最近微软DMTK团队在github上开源了性能超越其他boosting decision tree工具LightGBM，三天之内star了1000+次，fork了200+次。知乎上有近千人关注“如何看待微软开源的LightGBM？”问题，被评价为“**速度惊人**”，“**非常有启发**”，“**支持分布式**”，“**代码清晰易懂**”，“**占用内存小**”等。



且听来自MSRA开发团队的解读

Outline

有关GBDT的基础介绍

与已有系统的实验对比(e.g. XGBoost)

LightGBM中的优化

LightGBM的并行

动手指导与实践经验

Recall of supervised learning

- Component of supervised learning
  - Data: $[X, Y]$
    - $X = [x_1, x_2, \dots, x_n]^T, Y = [y_1, y_2, \dots, y_n]^T, x_i = [x_{i1}, x_{i2}, \dots, x_{im}]$
      - x_i is i-th training record, its label is y_i
      - x_{ij} is the j-th feature value of i-th training record.
  - Model/Function: F(X)
    - E.g. Linear model $F(x_i) = \sum_j w_j x_{ij}$
  - Loss Function: $L(F(X), Y) = \sum_i l(F(x_i), y_i)$
    - E.g. L2 loss: $l(F(x_i), y_i) = (F(x_i) - y_i)^2$
  - Regularization: $R(F(X))$, optional
    - E.g. L1 regularization on linear model: $R(F(X)) = \sum_j |w_j|$
- Goal of supervised learning
  - $F^*(X) = \arg\min_{F(X)}(L(F(X), Y) + R(F(x)))$

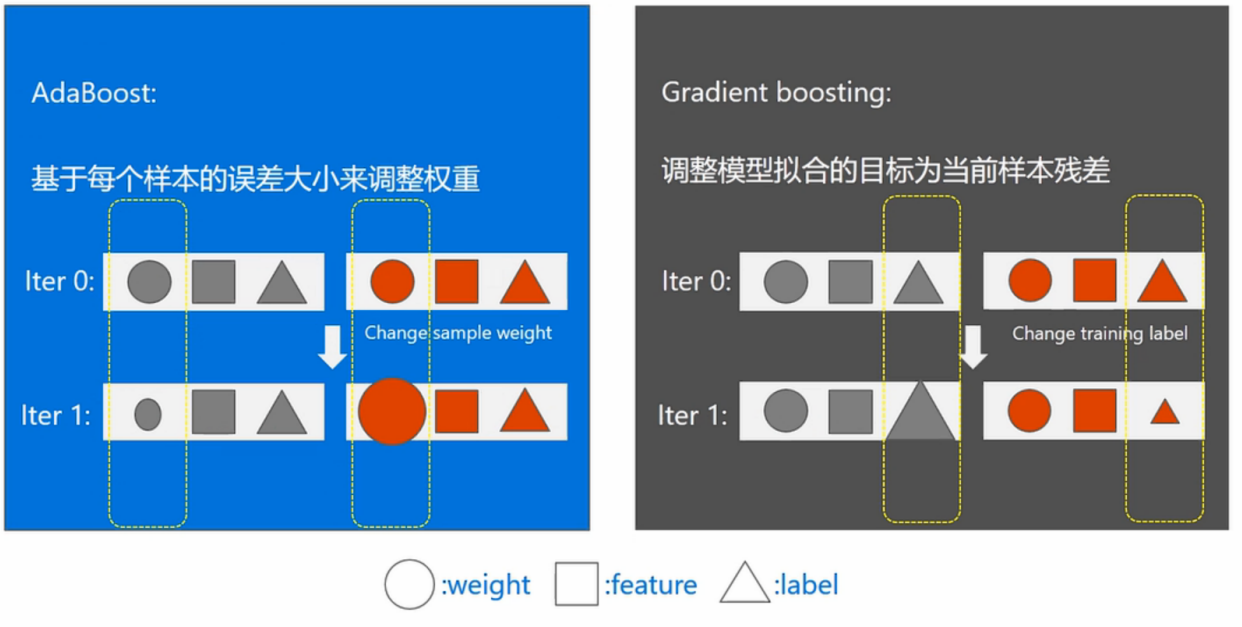
Loss function measures how well model fit on training data

Regularization measures complexity of model

Boosting

- You can view boosting as a linear combination of many models
  - $F_m(X) = \alpha_0 f_0(X) + \alpha_1 f_1(X) + \ ... + \alpha_m f_m(x)$
- It is stage-wise optimized algorithm
  - Learn F_0, then $F_1, F_2, ...$
  - Emphases error on each iteration
  - $L(F_m(X), Y) < L(F_{m-1}(X), Y)$
- AdaBoost
  - Emphases error by changing the distribution of samples
- Gradient Boosting
  - Emphases error by changing training targets.

Boosting


AdaBoost:
基于每个样本的误差大小来调整权重
Iter 0:
Change sample weight
Iter 1:
Gradient boosting:
调整模型拟合的目标为当前样本残差
Iter 0:
Change training label
Iter 1:
:weight
:feature
:label

Gradient Boosting

- Basic formulation
  - $F(X) = \sum_{m=0}^{M} f_m(X)$
  - $f(X)$ is "base learner", and we use decision tree in GBDT
- How to learn?
  - Greedy way
    - $F_m(X) = F_{m-1}(X) + f_m(X)$
    - Let $L(y, F_m(X)) < L(y, F_{m-1}(X))$
  - Gradient descent:
    - Get the negative gradient firstly
      - $\hat{y}_i = -\partial_{F_{m-1}(x_i)} l(F_{m-1}(x_i), y_i)$
    - Learn $f_m(X)$ to fit $\hat{Y}$ by using L2 loss
      - $f_m(X) = \arg\min_{f(X)} \sum_{i=1}^{n} (f(x_i) - \hat{y}_i)^2$

Thinking about L2 loss, $L = (y_i - F_{m-1}(x_i))^2$
then $\hat{y}_i = y_i - F_{m-1}(x_i)$

GBDT

- GBDT = Gradient Boosting + Decision Tree
- Use decision tree learner to fit gradients
- Supported tasks
  - Regression
  - Binary classification
    - pCTR task
  - Multi classification
  - Ranking
    - Lambdarank

Algorithm: **GBDT**

Input: Training data (X, Y) **, Iteration** M**, Loss function** l**, number of leaf C**

$F_0(X) = 0$

For m in (1,M):

▷ get the training targets

For all $(x_i, y_i) \in (X, Y)$:

$$y_i^m = -\partial_{F_{m-1}(x_i)} l\left(y_i, F_{m-1}(x_i)\right)$$

▷ use decision tree to fit targets

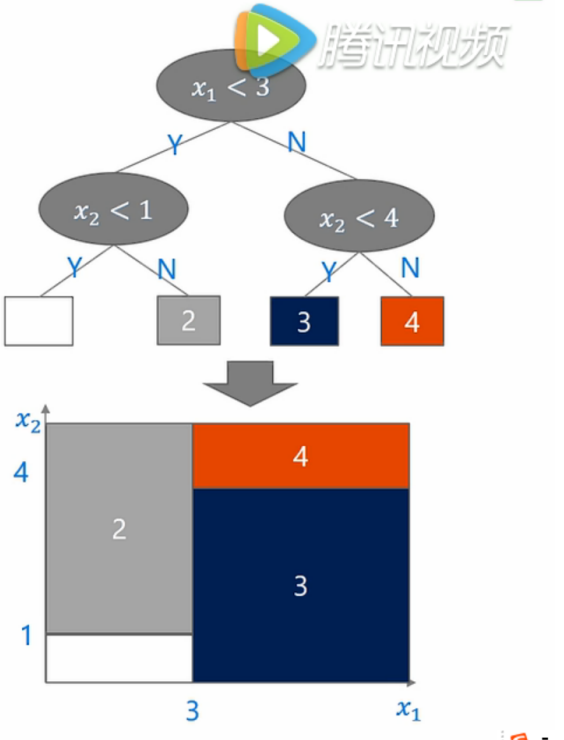
$f_m(X) = DecisionTree(X, Y^m, C, L2Loss)$

$F_m(X) = F_{m-1}(X) + f_m(X)$

Decision Tree

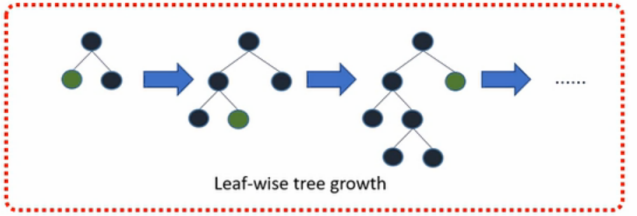
- Decision tree divided data into many non-overlapping regions
- Binary tree
- Components
  - Non-leaf node
    - Contain a simple decision rule(split), $\{feature, threshold\}$
    - Partition current region into two regions
  - Leaf node
    - Each x_i belongs one leaf
    - The data is same leaf means that data have similar label
    - Each leaf r_j has a output value
      - Generally, leaf output $w(r_j) = Average(\{y_i | i \in r_j\})$

- Formulation
  - $r_j = loc(x_i)$, get x_i's leaf index (x_i is a sample)
  - $T(X) = w(loc(X))$, *the output of the tree.*

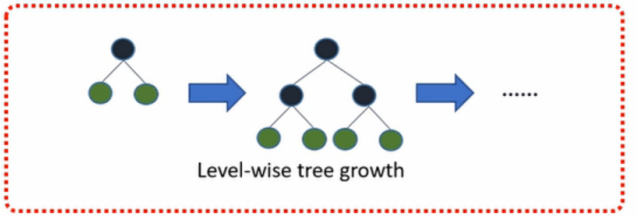


How to learn decision Tree

- Leaf-wise learning
  - Choose a leaf (with max delta loss)
  - split it into 2 leaves
  - $(p_m, f_m, v_m) = \arg\min_{(p,f,v)} L(T_{m-1}(X).split(p, f, v), Y)$
  - $T_m(X) = T_{m-1}(X).split(p_m, f_m, v_m)$



- Level-wise learning
  - split all leaves into 2 leaves



Decision Tree Learning Algorithm

Algorithm: **DecisionTree**
Input: Training data (X, Y) , number of leaf C, Loss function l
▷ put all data on root
$T_1(X) = X$
For m in (2,C):
▷ find best split
$(p_m, f_m, v_m) = FindBestSplit(X, Y, T_{m-1}, l)$
▷ perform split
$T_m(X) = T_{m-1}(X).split(p_m, f_m, v_m)$

Algorithm: **FindBestSplit**
Input: Training data (X, Y) , Loss function l, Current Model $T_{m-1}(X)$
For all Leaf p in $T_{m-1}(X)$:
For all f in X.Features:
For all v in f.Thresholds:
$(left, right) = partition(p, f, v)$
$\Delta loss = L(X_p, Y_p) - L(X_{left}, Y_{left}) - L(X_{right}, Y_{right})$
if $\Delta loss > \Delta loss(p_m, f_m, v_m)$:
$(p_m, f_m, v_m) = (p, f, v)$

Main computation cost in decision tree learning, can be further optimized

Existing Boosted Tree Tools

- XGBoost, pGBRT, Sklearn, R.GBM
  - XGBoost beat other tools on both training speed and accuracy[1]

Table 3: Comparison of Exact Greedy Methods with 500 trees on Higgs-1M data.

Method	Time per Tree (sec)	Test AUC
XGBoost	0.6841	0.8304
XGBoost (colsample=0.5)	0.6401	0.8245
scikit-learn	28.51	0.8302
R.gbm	1.032	0.6224

Table 4: Comparison of Learning to Rank with 500 trees on Yahoo! LTRC Dataset

Method	Time per Tree (sec)	NDCG@10
XGBoost	0.826	0.7892
XGBoost (colsample=0.5)	0.506	0.7913
pGBRT [22]	2.576	0.7915

[1] Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." *arXiv preprint arXiv:1603.02754* (2016).

Implementation of XGBoost

- Pre-sorted(exact) algorithm
  - Pros:
    - Can find exact split point
  - Cons:
    - High computation cost
    - High memory usage
    - Exact split point may cause over-fitting
- Level-wise tree growth
  - Pros:
    - Easy for multi-threading optimization
    - Can speed up pre-sorted algorithm
  - Cons:
    - Not efficient, may grow unnecessary leaf

LightGBM

- LightGBM is a gradient boosting framework. It is designed to be distributed and efficient with following advantages:
  - Fast training speed and high efficiency
  - Lower memory usage
  - Better accuracy
  - Parallel learning supported
  - Capability of handling large-scaling data
  - Support categorical feature directly
- https://github.com/Microsoft/LightGBM

What we do in LightGBM ?

	XGBoost	LightGBM
Tree growth algorithm	Level-wise good for engineering optimization but not efficient to learn model	Leaf-wise with max depth limitation get better trees with smaller computation cost, also can avoid overfitting
Split search algorithm	Pre-sorted algorithm	Histogram algorithm
memory cost	2*#feature*#data*4Bytes	#feature*#data*1Bytes (8x smaller)
Calculation of split gain	O(#data* #features)	O(#bin *#features)
Cache-line aware optimization	n/a	40% speed-up on Higgs data
Categorical feature support	n/a	8x speed-up on Expo data

Comparison

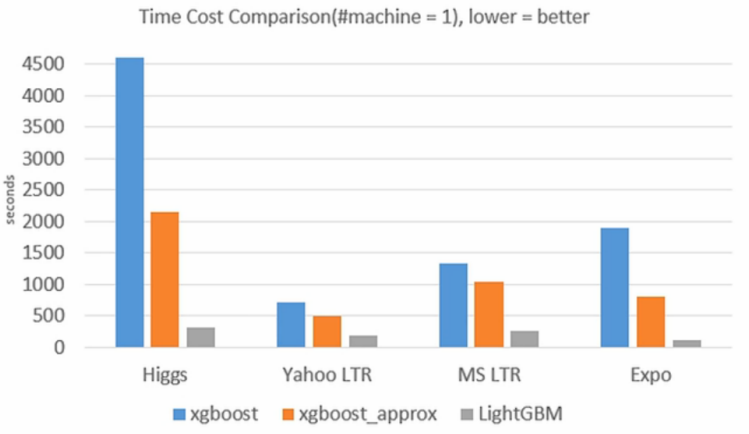
Accuracy:

Data	Metric	xgboost	xgboost_approx	LightGBM
Higgs	AUC	0.839528	0.840533	0.845123
Yahoo LTR	NDCG@5	0.740316	0.739363	0.756369
MSLTR	NDCG@5	0.476245	0.474441	0.510153
Expo	AUC	0.75548	0.757071	0.781061

Memory consumption:

Data	xgboost	xgboost_approx	LightGBM
Higgs	4.853GB	4.875GB	0.822GB
Yahoo LTR	1.907GB	2.221GB	0.831GB
MS LTR	5.469GB	5.600GB	0.745GB
Expo	1.553GB	1.560GB	0.450GB

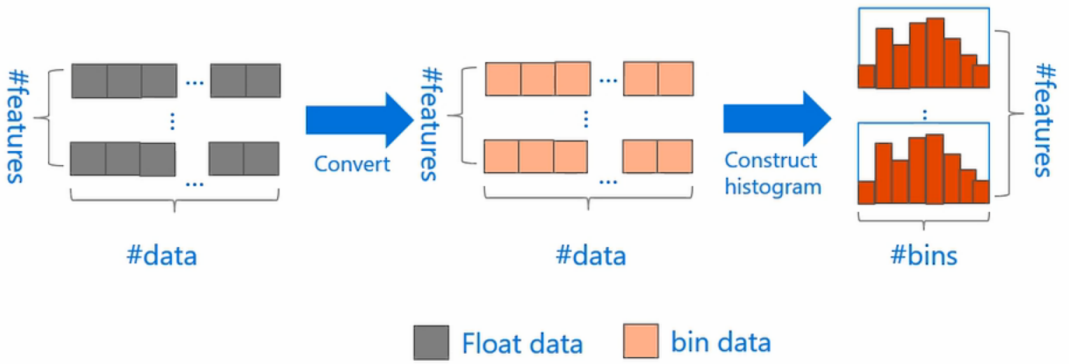
Speed:



Detail Optimizations

Histogram optimization

- Compression of feature
- Map continues values to discrete values(called "bin")
  - E.g. [0,0.1) -> 0, [0.1,0.3)->1, ...



Histogram optimization

Convert feature value to bin before training

Use bin to index histogram, not need to sort

Reduce computation cost of split gain

Algorithm: **FindBestSplitByHistogram**

Input: Training data X **, Current Model** $T_{c-1}(X)$

First order gradient G, second order gradient H

For all Leaf p in $T_{c-1}(X)$:

For all f in X.Features:

▷ construct histogram

H = new Histogram()

For i in (0, num_of_row) //go through all the data row

$H[f.bins[i]].g \mathrel{+}= g_i; H[f.bins[i]].n \mathrel{+}= 1$

▷ find best split from histogram

For i in (0,len(H)): //go through all the bins

$S_L \mathrel{+}= H[i].g; n_L \mathrel{+}= H[i].n$

$S_R = S_P - S_L; n_R = n_P - n_L$

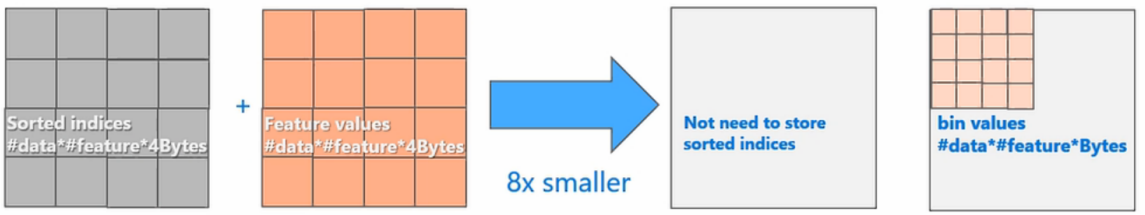
$\Delta loss = \frac{S_L^2}{n_L} + \frac{S_R^2}{n_R} - \frac{S_P^2}{n_P}$

if $\Delta loss > \Delta loss(p_m, f_m, v_m)$:

$(p_m, f_m, v_m) = (p, f, H[i].value)$

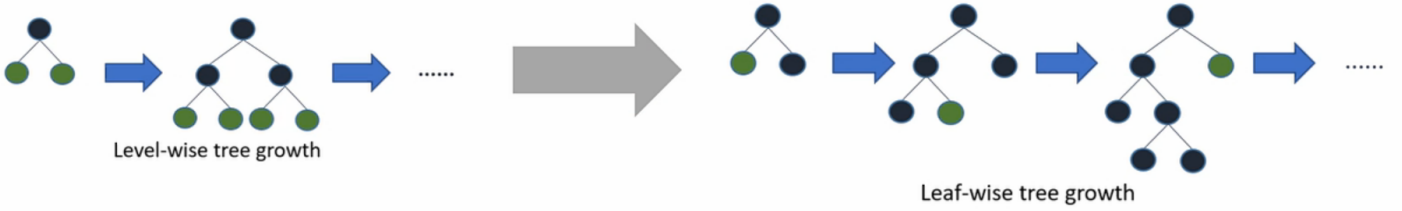
Memory usage optimization

- Only need to save bin values.
- If #bins is small, can use small data type, e.g. uint8_t, to store training data



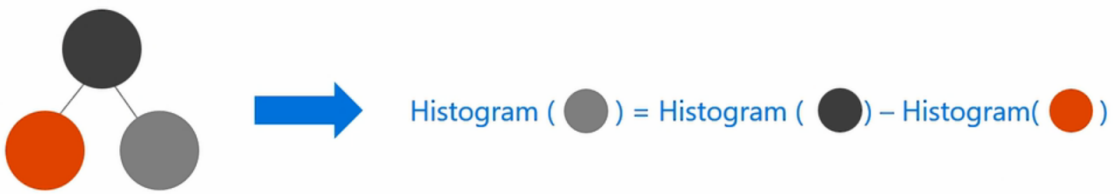
Leaf-wise with max depth limitation

- Use leaf-wise to grow trees efficiently
  - Can reduce more loss when growing same #leaves
- Use max depth to avoid growing too deep tree and overfitting



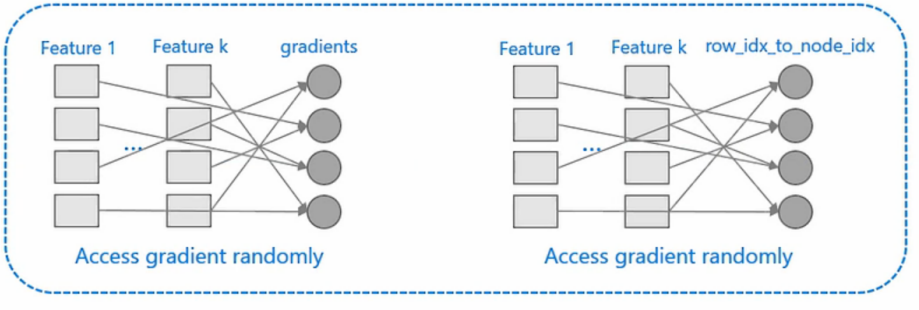
Histogram subtraction

- To get one leaf's histograms in a binary tree, can use the histogram subtraction of its parent and its neighbor
- Construct histograms for one leaf (with smaller #data than its neighbor), then can get histograms of its neighbor by histogram subtraction with small cost(O(#bins))
- 2x speed-up



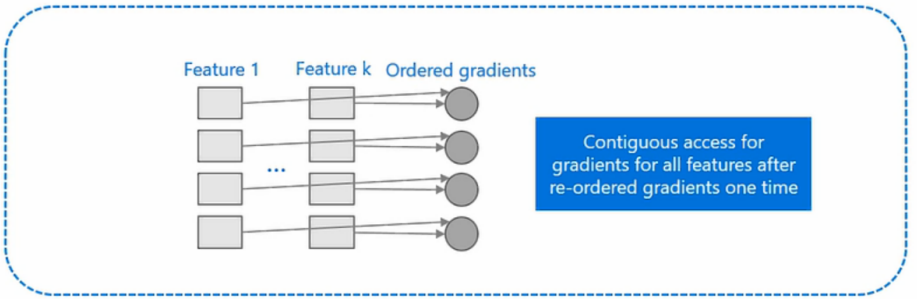
Increase cache hit chance

- Pre-sorted algorithm will suffer from cache-miss
  - Access gradients randomly, O(#data * #feature)
    - Different feature will access gradients by different order
  - Access row_idx_to_node_idx randomly, O(#data * #feature)
    - Pre-sorted algorithm use row_idx_to_node_idx to speed up “split” procedure
    - But it access this array randomly
- These random access cause many cache miss



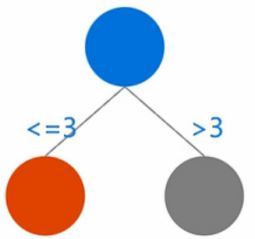
Increase cache hit chance

- Histogram based algorithm is easy to increase cache hit chance
  - Access gradient continuous
    - Different features access gradients by same order
    - re-order gradients one time, and every feature can access gradients contiguously
  - Not need to use row_idx_to_node_idx array

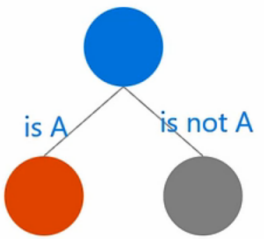


Categorical feature support

- Existing tools use one-hot coding to support categorical feature
  - Need expand feature dimension. Cost much on time and space
- LightGBM can direct use categorical feature as input
  - Change decision rule for categorical features



Numerical feature decision rule

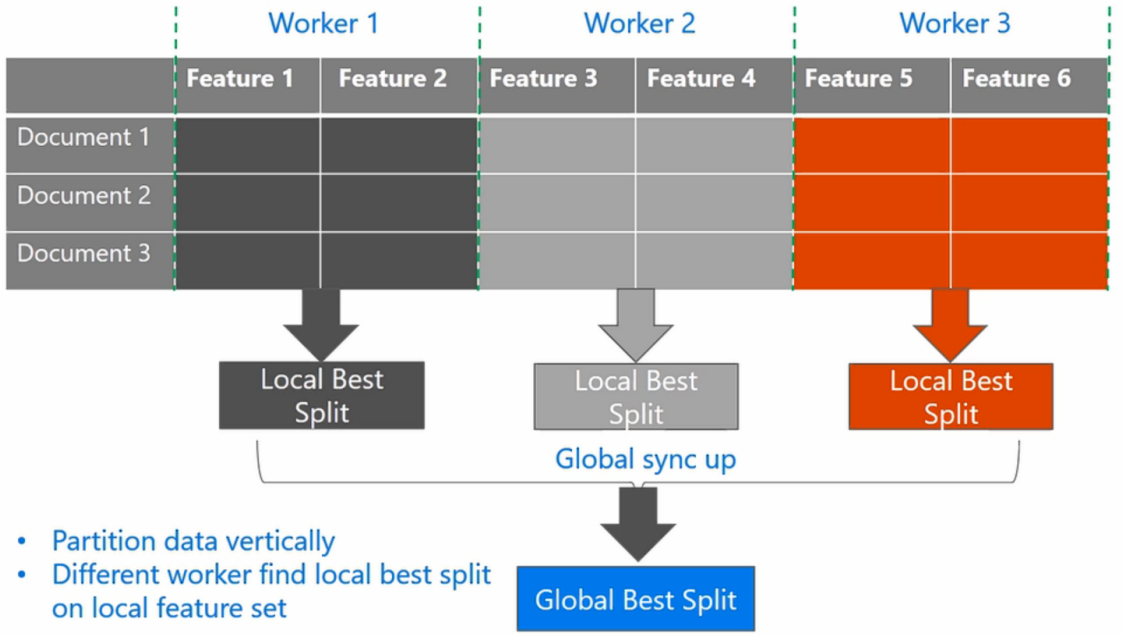


Categorical feature decision rule

Parallel Learning Support

- Feature Parallelization
  - Cost of computation and communication are depending on #data
- Data Parallelization
  - Communication cost is depending on #features
- Voting Parallelization
  - Reduce the communication in data parallelization
    - Constant communication cost
  - Accepted by NIPS 2016

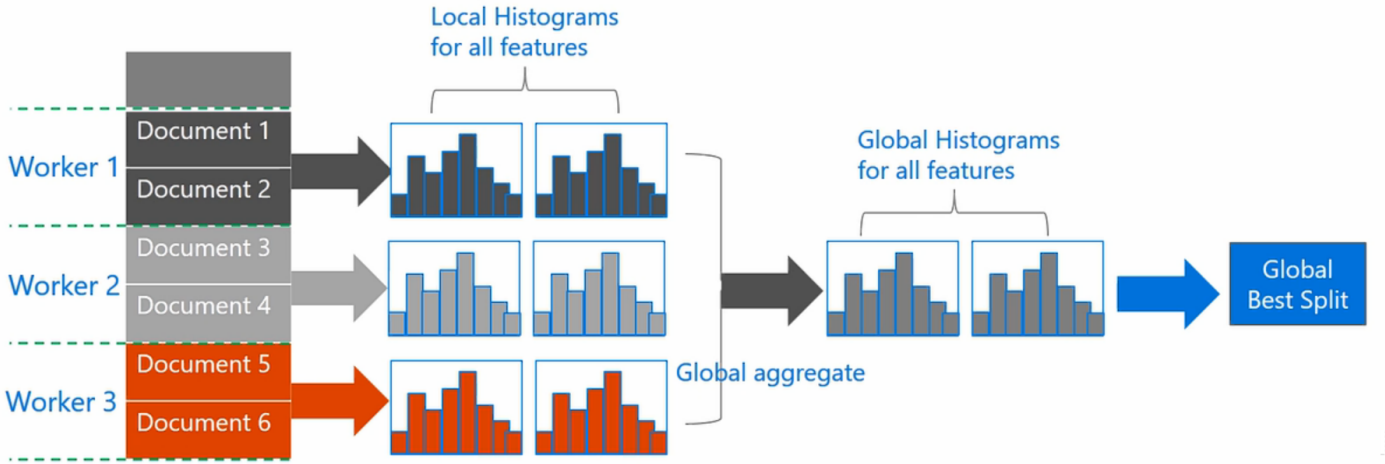
Feature/Attribute Parallelization



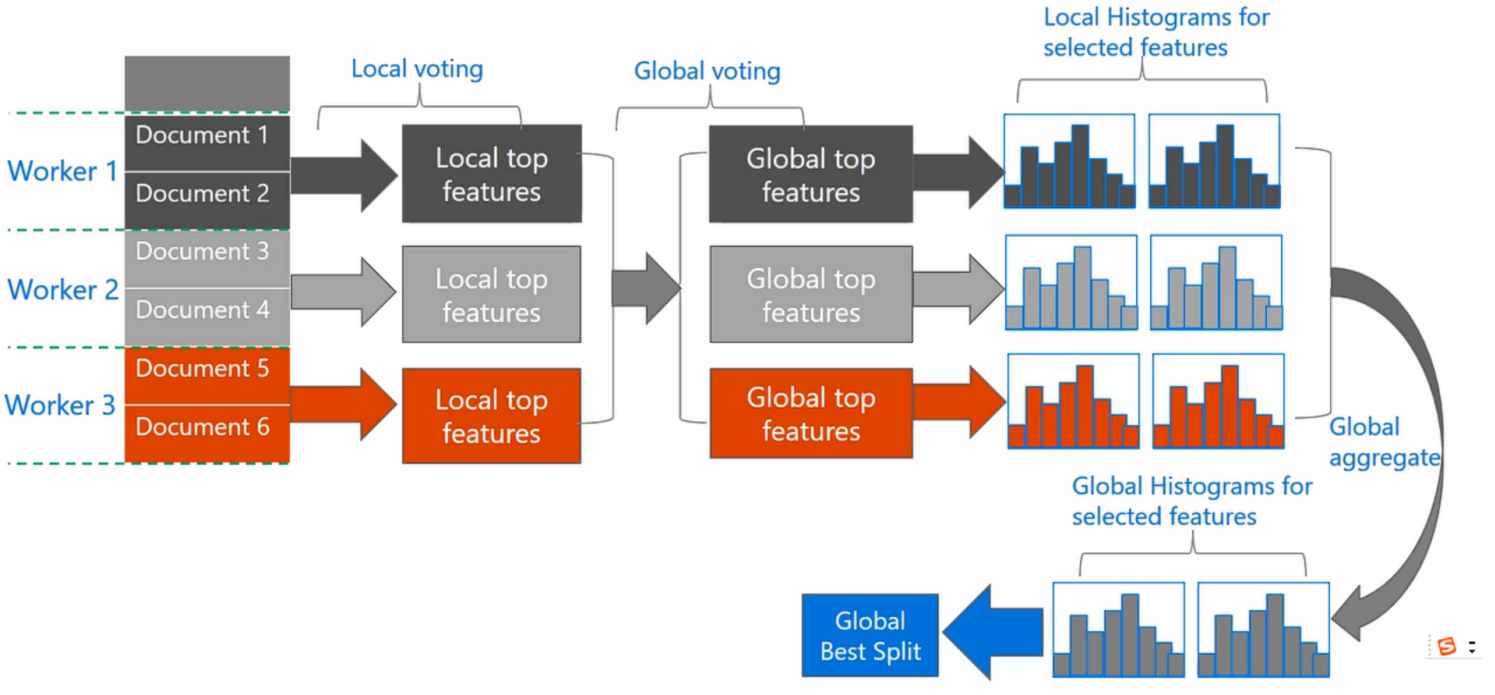
- Partition data vertically
- Different worker find local best split on local feature set

Data Parallelization

- Partition data horizontally
- Each worker construct local histograms, then aggregate global histograms from all local histograms



Parallel optimization – voting based parallel



Parallel Experiments

- learning to rank task, about 11M training data

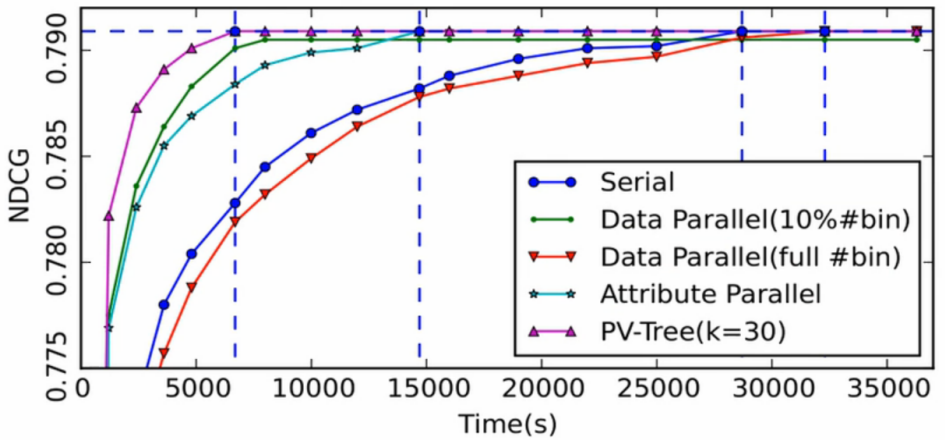


Figure 1. Performances of different algorithms (LTR, 8 machines)

Parallel Experiments

- pCTR task, about 235M training data

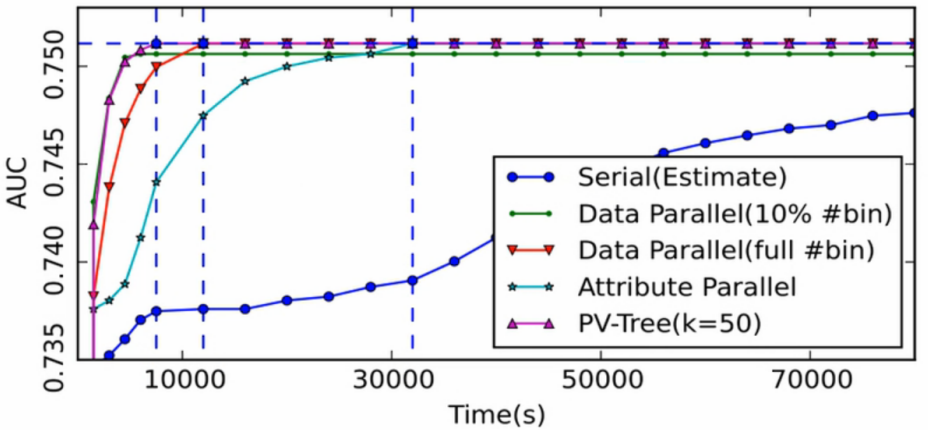


Figure 2. Performances of different algorithms (CTR, 32 machines)

Others Features

Model training tricks	Efficiency	User-friendly functions
• Regularization methods • DART(drop out) support • Weighted training • Sub-samples (Bagging and sub-feature)	• Optimize for sparse features • Multi-threading optimization	• Continued train • Early stopping • Cross Validation • Python support

Command line quick start

- Prepare your data in Libsvm, csv or tsv format
- train it with lightgbm

```
lightgbm.exe data=train.svm valid=valid.svm num_trees=50 num_leaves=100 output_model=model.txt
```

- Parallel training

```
mpiexec lightgbm.exe num_machine=2 tree_learner=voting data=train.svm
valid=valid.svm num_trees=50 num_leaves=100 output_model=model.txt
```

- After trained, you can get the model file: “model.txt”, use it to predict

```
lightgbm.exe task=prediction data=test.svm input_model=model.txt
```

Python Quick start guide

```python
import lightgbm as lgb
import pandas as pd
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV

# load or create your dataset
print('Load data...')
df_train = pd.read_csv('../regression/regression.train', header=None, sep='\t')
df_test = pd.read_csv('../regression/regression.test', header=None, sep='\t')

y_train = df_train[0]
y_test = df_test[0]
X_train = df_train.drop(0, axis=1)
X_test = df_test.drop(0, axis=1)

# train
gbm = lgb.LGBMRegressor(objective='regression',
                        num_leaves=31,
                        learning_rate=0.05,
                        n_estimators=20)
gbm.fit(X_train, y_train,
        eval_set=[(X_test, y_test)],
        eval_metric='l1',
        early_stopping_rounds=5)

# predict
y_pred = gbm.predict(X_test, num_iteration=gbm.best_iteration)
# eval
print('The rmse of prediction is:', mean_squared_error(y_test, y_pred) ** 0.5)
```

Convert max_depth to num_leaves

- num_leaves = 2^(max_depth)

max_depth	num_leaves
1	2
2	4
3	8
7	128
10	1024

Faster learning speed

- Bagging (data sub-sampling)
- Feature sub-sampling
- Use categorical feature directly
- Save data file to binary file to speed up data loading in future learning
- Use parallel learning

For better accuracy

- Small learning_rate with large num_iterations
- Large num_leave(may over-fitting)
- Cross validation
- Bigger training data
- Try DART – use drop out during the training

Deal with overfitting

- small max_bin – feature - 分桶略微粗一些
- small num_leaves – 不要在单棵树上分的太细
- Control min_data_in_leaf and min_sum_hessian_in_leaf – 确保叶子节点还有足够多的数据
- Sub-sample – 在构建每棵树的时候，在data上做一些sample
- Sub-feature – 在构建每棵树的时候，在feature上做一些sample
- bigger training data – 更多的训练数据
- lambda_l1, lambda_l2 and min_gain_to_split to regularization -正则
- max_depth to avoid growing deep tree – 控制树深度

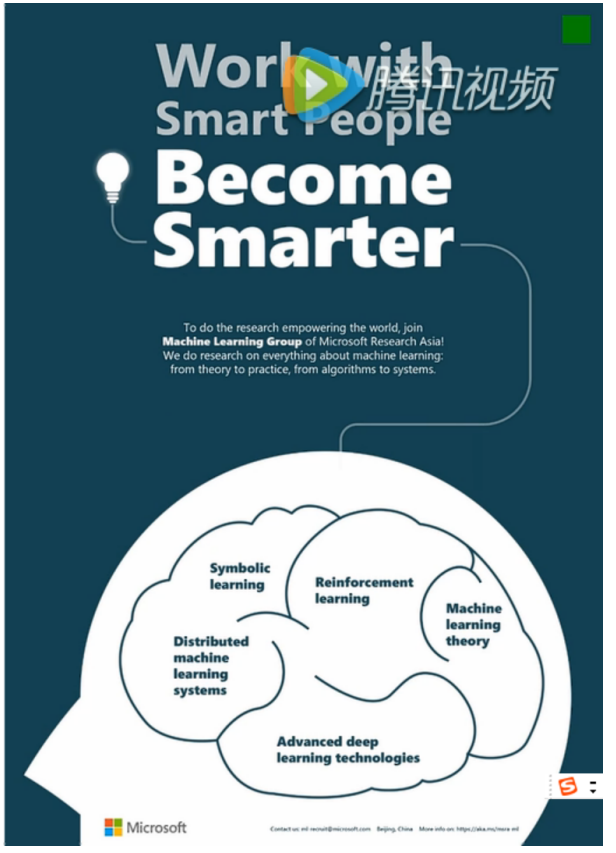
- Star LightGBM in Github:
  https://github.com/Microsoft/LightGBM
- Open issues when met problems:
  https://github.com/Microsoft/LightGBM/issues
- Welcome to contribute:
  https://github.com/Microsoft/LightGBM/issues?q=is%3Aissue+is%3Aopen+label%3Acall-for-contribution

Looking forward

- Speed up with GPU
- More language support – R/Scala/Julia
- More platform support – Hadoop/Spark
- Online version – learn from infinite data
- Integration/Fusion with DNN

We are hiring!

Contact us by: tyliu@microsoft.com



MSRA WeMedia Platforms

腾讯视频

头条

一点资讯

搜狐 SOHU.com

WeChat: 微软研究院AI头条

- Top 10 most popular AI WeChat account

Weibo: 微软亚洲研究院

- http://weibo.com/msra

Toutiao.com: 微软亚洲研究院

- http://www.toutiao.com/m6088543147/

Yidianzixun.com: 微软研究院

- http://www.yidianzixun.com/home?page=channel&id=m140441

Sohu.com: 微软亚洲研究院

- http://mp.sohu.com/profile?xpt=MTMwODIzNTg0M0BzaW5hLnNvaHUuY29t

参考资料

[1].LightGBM: A Highly Efficient Gradient Boosting Decision Tree

[2].lightGBM视频教程

[3].lightGBM-CN

[4].论文解读1

[5].论文解读2

[6].论文解读3