# LightGBM

- Installation-Guide : https://lightgbm.readthedocs.io/en/latest/Installation-Guide.html
- Documeatation : https://media.readthedocs.org/pdf/testlightgbm/latest/testlightgbm.pdf

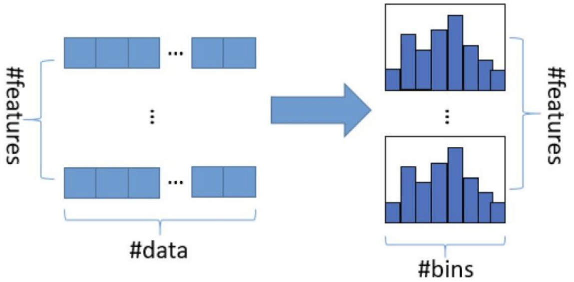
# XGBoost v.s. LightGBM

- Comparison (http://wepon.me/files/gbdt.pdf)

## LightGBM的改进

- 直方图算法

把连续的浮点特征值离散化成k个整数，同时构造一个宽度为k的直方图。在遍历数据的时候，根据离散化后的值作为索引在直方图中累积统计量，当遍历一次数据后，直方图累积了需要的统计量，然后根据直方图的离散值，遍历寻找最优的分割点




- 减小内存占用，比如离散为256个bin时，只需要8bit，节省7/8
- 减小了split finding时计算增益的计算量， 从O(#data) 降到O(#bins)

# Data Preparation for LightGBM

LightGBM can use categorical features as input directly. It doesn’t need to convert to one-hot coding, and is much faster than one-hot coding (about 8x speed-up).

> *Note: You should convert your categorical features to int type before you construct Dataset for LGBM. It does not accept string values even if you passes it through categorical_feature parameter.*

## 2. Data Preparation for LightGBM

In [17]:
```
application_train = pd.read_csv('../Kaggle data/application_train.csv')

# use LabelEncoder to convert categorical features to int type before construct Dataset
from sklearn.preprocessing import LabelEncoder
def label_encoder(input_df, encoder_dict=None):
    """ Process a dataframe into a form useable by LightGBM """
    # Label encode categoricals
    categorical_feats = input_df.columns[input_df.dtypes == 'object']
    for feat in categorical_feats:
        encoder = LabelEncoder()
        input_df[feat] = encoder.fit_transform(input_df[feat].fillna('NULL'))
    return input_df, categorical_feats.tolist(), encoder_dict
application_train, categorical_feats, encoder_dict = label_encoder(application_train)
X = application_train.drop('TARGET', axis=1)
y = application_train.TARGET
```

# Tuning Hyperparameters

- **Article**
  - https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc
  - https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/

## For Faster Speed

- Use bagging by setting `bagging_fraction` and `bagging_freq`
- Use feature sub-sampling by setting `feature_fraction`
- Use small `max_bin`
- Use `save_binary` to speed up data loading in future learning
- Use parallel learning, refer to Parallel Learning Guide

## For Better Accuracy

- Use large `max_bin` (may be slower)
- Use small `learning_rate` with large `num_iterations`
- Use large `num_leaves` (may cause over-fitting)
- Use bigger training data
- Try `dart`

## Deal with Over-fitting

- Use small `max_bin`
- Use small `num_leaves`
- Use `min_data_in_leaf` and `min_sum_hessian_in_leaf`
- Use bagging by set `bagging_fraction` and `bagging_freq`
- Use feature sub-sampling by set `feature_fraction`
- Use bigger training data
- Try `lambda_l1`, `lambda_l2` and `min_gain_to_split` for regularization
- Try `max_depth` to avoid growing deep tree

Your most recent submission

| Name | Submitted | Wait time | Execution time | Score |
|---|---|---|---|---|
| submissions_toy_grid.csv | just now | 1 seconds | 0 seconds | 0.721 |

Complete

Jump to your position on the leaderboard

# Demo

Data: https://www.kaggle.com/c/home-credit-default-risk/

Your most recent submission

| Name | Submitted | Wait time | Execution time | Score |
|---|---|---|---|---|
| submissions_toy_bayes.csv | just now | 1 seconds | 0 seconds | 0.738 |

Complete

Jump to your position on the leaderboard

# Tuning Hyperparameters - Two Methods

- **Grid Search**
  - from sklearn.model_selection import GridSearchCV
- **Bayesian Optimization**
  (https://www.kaggle.com/sz8416/simple-bayesian-optimization-for-lightgbm)
  - from bayes_opt import BayesianOptimization