Apache Flink

基于Flink的高性能机器学习算法库

杨旭 · 阿里巴巴计算平台事业部 / 资深算法专家

Flink Meetup 北京 – 2019年06月29日

Apache Flink

CONTENT

目录 >>

Alink 支持的数据源

Apache Flink

数据源	Batch读	Batch写	Stream读	Stream写
ODPS	√	√	√	√
DataHub			√	√
TT			√	√
MetaQ			√	√
SLS			√	
Tair				√
Swift			√	√
Notify			√	
MySql	√	√	√	√
AliHBase	√	√	√	√
Tddl（idb）	√	√	√	√
MongoDB	√		√	
CSV	√	√	√	√
SQLite	√	√	√	√
Derby	√	√	√	√

Alink算法功能列表（1/4）

- 回归
  - 线性回归, Lasso, Ridge,支持向量回归, Stepwise, Cart, GBDT, 随机森林
- 分类
  - 逻辑回归, SVM, 感知机, 朴素贝叶斯, KNN, Tradaboost, 随机森林, ID3, Cart, C45
- 聚类
  - KMeans, KModes, DBSCan, AGNES, PIC
- 深度学习
  - DL模型训练和预测, TensorFlow预测
- 在线学习
  - FTRL，KMeans， Perceptron，Passive Aggressive (PA)，PA-I，PA-II
- 评估
  - 分类评估, 聚类评估

Alink算法功能列表（2/4）

- 数据处理
  - 随机采样，分层采样
  - 归一化，标准化，缺失值填充，类型转换
  - KvToTensor,TableToTensor,TensorToTable,TensorFunction,TensorToTuples
  - Velocity变量，网络流量指标，TensorExpandDim, appendId(batch)
  - 单列拆分成多行, 列拆分后选取, Json值抽取, 单列拆分成多列, 多列拼接为单行
  - SqlCmd,As,Select,UnionAll,Where,GroupBy(batch),Distinct(batch),
  - Intersect(batch),Join(batch),Minus(batch),Orderby(batch)
  - 多流合并，LatestJoin, Lookup

- 特征工程
  - onehot编码预测，特征尺度变换, 特征异常平滑，线性模型特征重要性分析

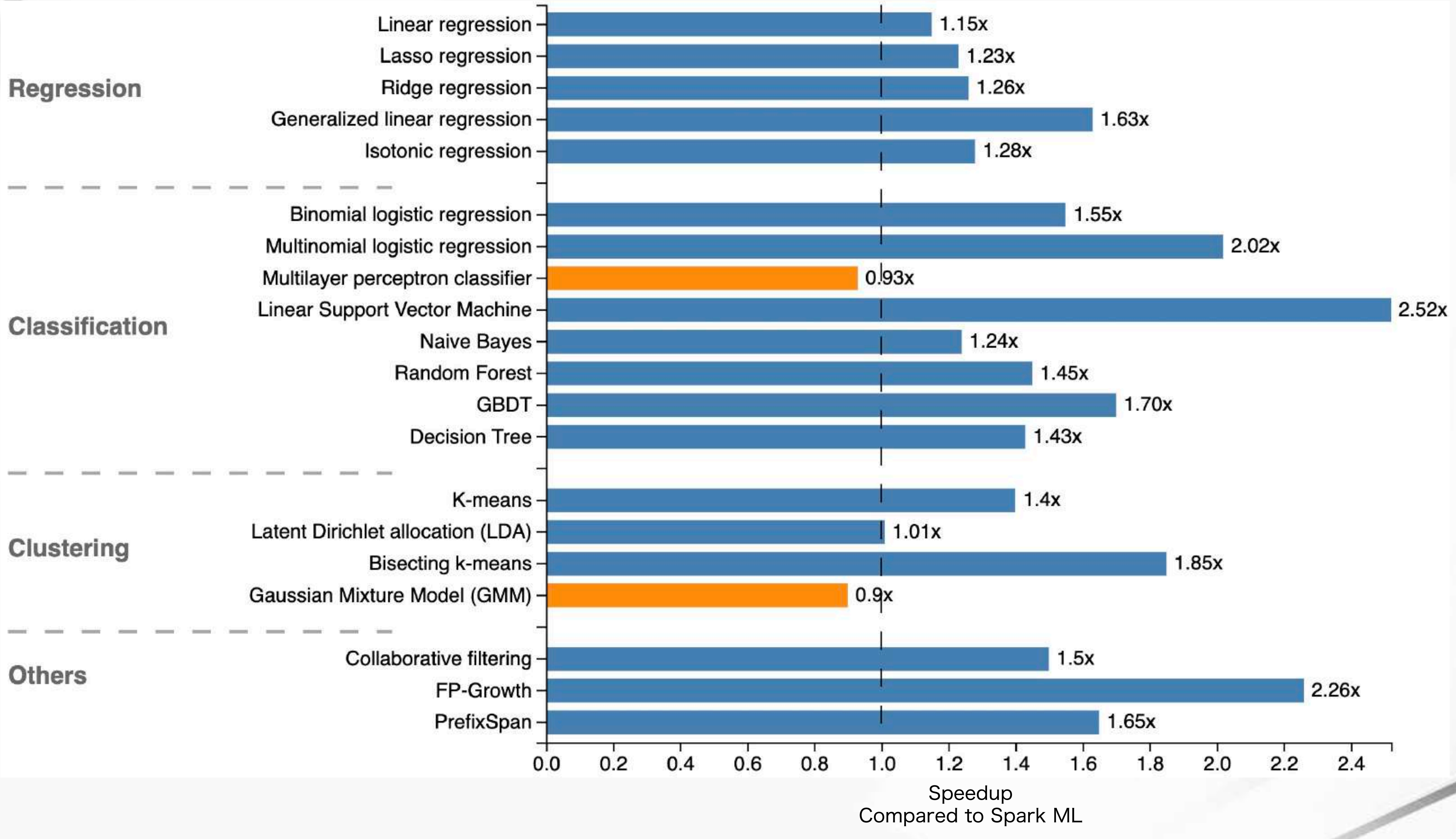
Alink算法功能列表（3/4）

- 基本统计
  - 窗口统计，全表统计，分组窗口统计
  - 个数,求和,均值,最大值,最小值,缺失值个数,方差, 标准差,标准误, 峰度, 偏度等
  - 最大的k个值，最小的k个值
- 变量关系
  - 相关系数，协方差，对应分析，交叉表，多重共线性
- 数据分布
  - 百分位，频率，直方图，概率密度图，累计密度图，pp图，洛伦兹曲线
- 假设检验
  - T检验，chi2检验，AD检验，KS检验
- 数据降维
  - 主成分分析，tSNE
- 时间序列
  - ARIMA, Garch, ArimaGarch

Alink算法功能列表（4/4）

- 异常检测
  - SOS, KSigma, AVF, Boxplot, AGD, OneClassSvm, SMA, EWMA, CDM, G检验 UriNumberDetection, GroupDetection, GroupMFIDetection，BigGraphGeneration
- 推荐算法
  - ALS，Simrank，FM, ItemCF
- 文本
  - 词频统计，分词，停用词过滤，Tokenizer, 新词识别 (batch), TFIDF(batch) ，文本特征生成
  - Word2Vec(batch), 文本敏感数字抓取，银行卡信息解析，身份证信息解析，单词序列转ID序列， 字符串相似度，文本相似度, 语义向量距离(batch)，SimHash
- 图算法
  - 单源最短路径, 社区发现, 标签传播, PageRank, HITS, 树深度算法, 连通图
  - 模块度, Kcore, 三角形数目统计, 二度邻居查找

Major ML Algorithms



FM 算法

因子分解机 (Factorization Machine, FM) 是由 Steffen Rendle 2010 年提出的一种基于矩阵分解的机器学习算法，常用于大规模的CTR预估； 其主要优点包括:

- 可用于高度稀疏数据场景；
- 具有线性的计算复杂度。

- **线性模型**

$$y = w_0 + \sum_{i=1}^{n} w_i x_i$$

- **二阶多项式模型**

$$y = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{ij} x_i x_j$$

- **因子分解机(FM)模型**

$$y = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} < v_i, v_j > x_i x_j$$

ALGORITHM 1: Stochastic Gradient Descent (SGD)

Input: Training data S, regularization parameters λ, learning rate η, initialization σ
Output: Model parameters $\Theta = (w_0, \mathbf{w}, \mathbf{V})$
$w_0 \leftarrow 0$; $\mathbf{w} \leftarrow (0, \ldots, 0)$; $\mathbf{V} \sim \mathcal{N}(0, \sigma)$;
repeat
  for $(\mathbf{x}, y) \in S$ **do**
    $w_0 \leftarrow w_0 - \eta \left(\frac{\partial}{\partial w_0} l(\hat{y}(\mathbf{x}|\Theta), y) + 2\lambda^0 w_0 \right)$;
    for $i \in \{1, \ldots, p\} \wedge x_i \neq 0$ **do**
      $w_i \leftarrow w_i - \eta \left(\frac{\partial}{\partial w_i} l(\hat{y}(\mathbf{x}|\Theta), y) + 2\lambda^w_{\pi(i)} w_i \right)$;
      for $f \in \{1, \ldots, k\}$ **do**
        $v_{i,f} \leftarrow v_{i,f} - \eta \left(\frac{\partial}{\partial v_{i,f}} l(\hat{y}(\mathbf{x}|\Theta), y) + 2\lambda^v_{f,\pi(i)} v_{i,f} \right)$;
      end
    end
  end
until *stopping criterion is met*;

Graph Embedding

- Huge Graph from Industry

  Facebook: ~2 billion active users

  Wechat: ~1 billion active users

  Amazon: 400M active users, 400M products

  Taobao: 500M active users, 800M products

- Alink supported algorithms with billions of nodes

  1、DeepWalk

  2、Node2Vec

  3、MetaPath2Vec

阿里巴巴基于Flink的通用算法平台——Alink

- PAI 算法平台的一部分，是基于 Flink 的算法平台。
  - 该平台希望通过提供丰富的算法库及便捷的编辑运行环境，帮助数据分析和应用开发人员快速高效的实现各种批/流数据的分析和处理。
- 核心是丰富的数据分析算法库
  - 包含常用统计分析、机器学习、文本处理、推荐、异常检测等多个领域的算法
- 覆盖数据分析和建模的全流程
  - 数据分析和应用开发人员能够从数据探索、模型训练、实时预测、可视化展示，端到端地完成整个流程。

Alink 名称的由来

➢相关名称的公共部分

- Alibaba, Algorithm, AI, Flink, Blink

➢各算法功能通过“link”的方式进行链接

- Web UI 工作流的各个节点是通过连线，连接（link）起来
- 在编程调用时，每个算子都定义了link方法

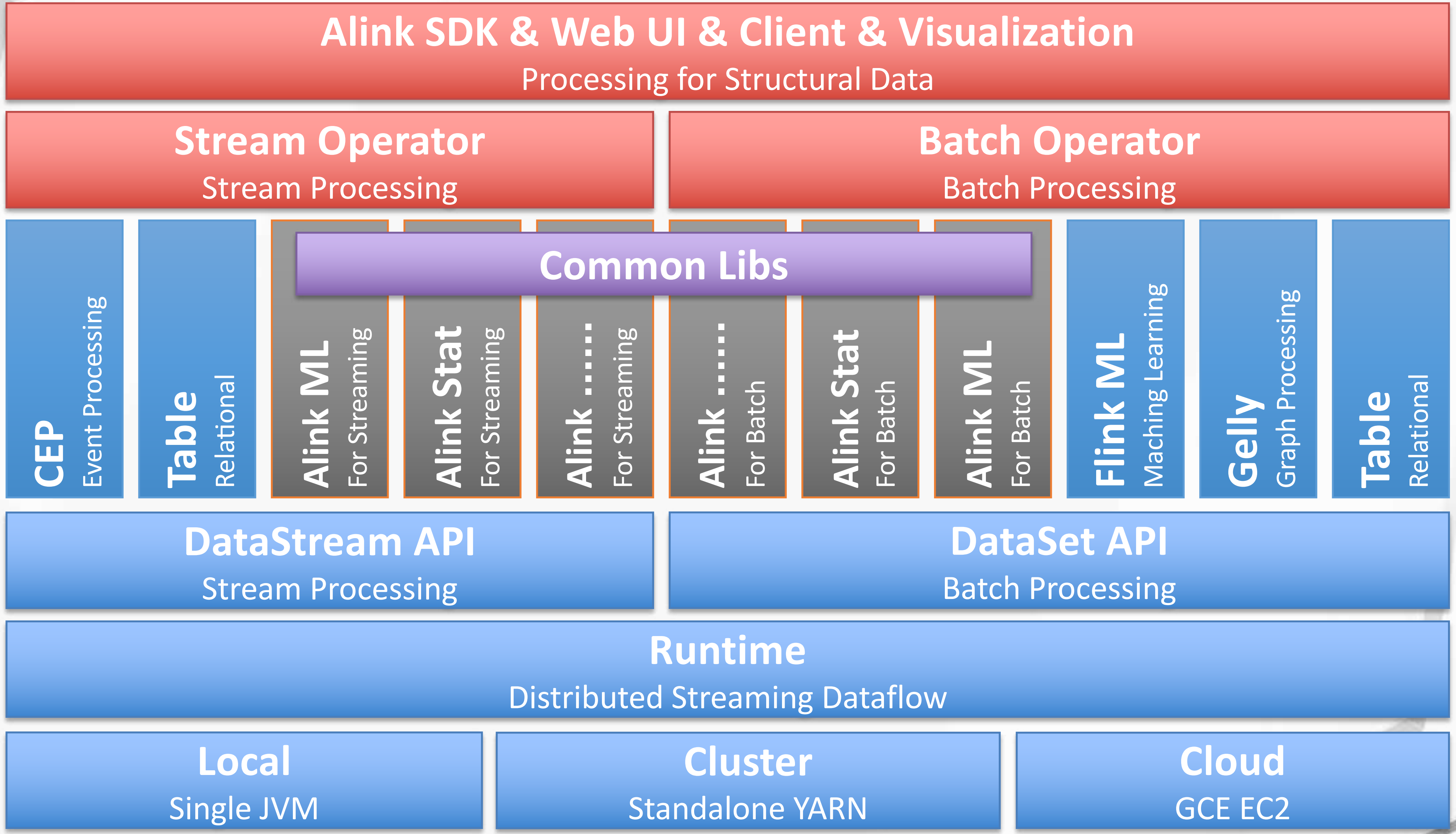
```
op1.link(op2)
```

```
op3.linkFrom(op1,op2)
```

- 自动处理相关的meta等信息，便于用户方便快捷搭建业务流程

Alink 架构

Apache Flink



Alink 如何使用？

➢ 网页前端

- 即当前PAI Web所提供的，通过拖拽、配置算法组件就可完成业务逻辑的描述，流式算法的组件的使用方式上与批处理的组件相同，学习门槛低，能够快速上手。

➢ PC客户端

- 提供了本地运行的功能，对于小规模的数据，可以直接在个人的台式机或笔记本上进行快速的分析和处理
- 提供了脚本编辑运行功能，通过脚本进行复杂的流程控制；脚本可以在本地运行，或提交到集群运行

➢ 命令行

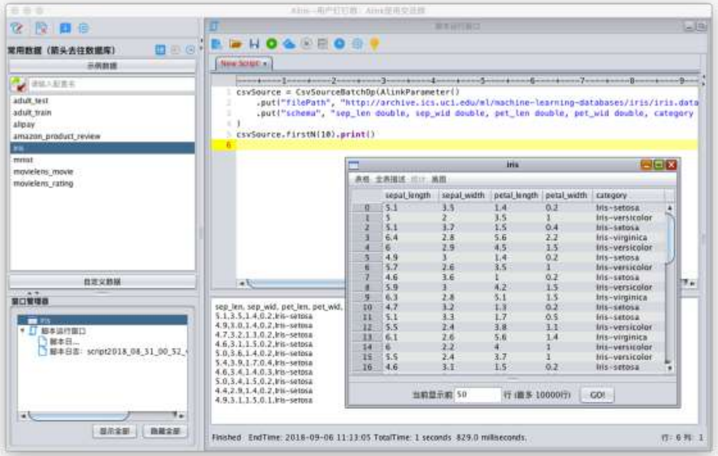
- 运行Alink脚本，可以在本地运行，也可以提交到集群运行

Alink 如何使用？

- 多种调用方式，适合不同用户及场景

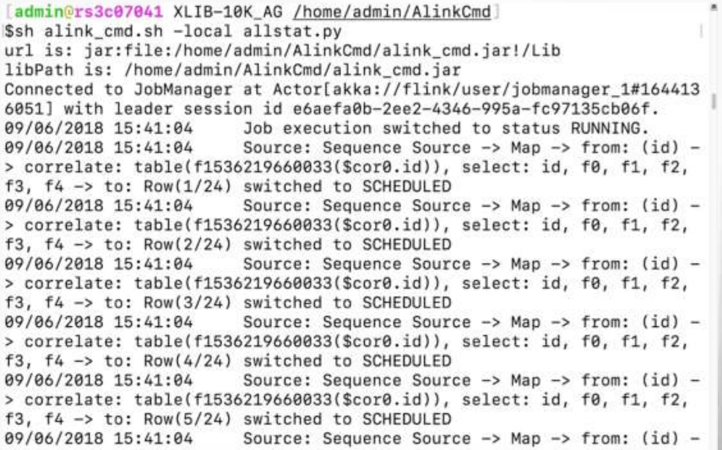
网页前端

简单便捷，工作流配置和执行



PC客户端

支持脚本编辑运行，支持本地运行与集群运行



命令行

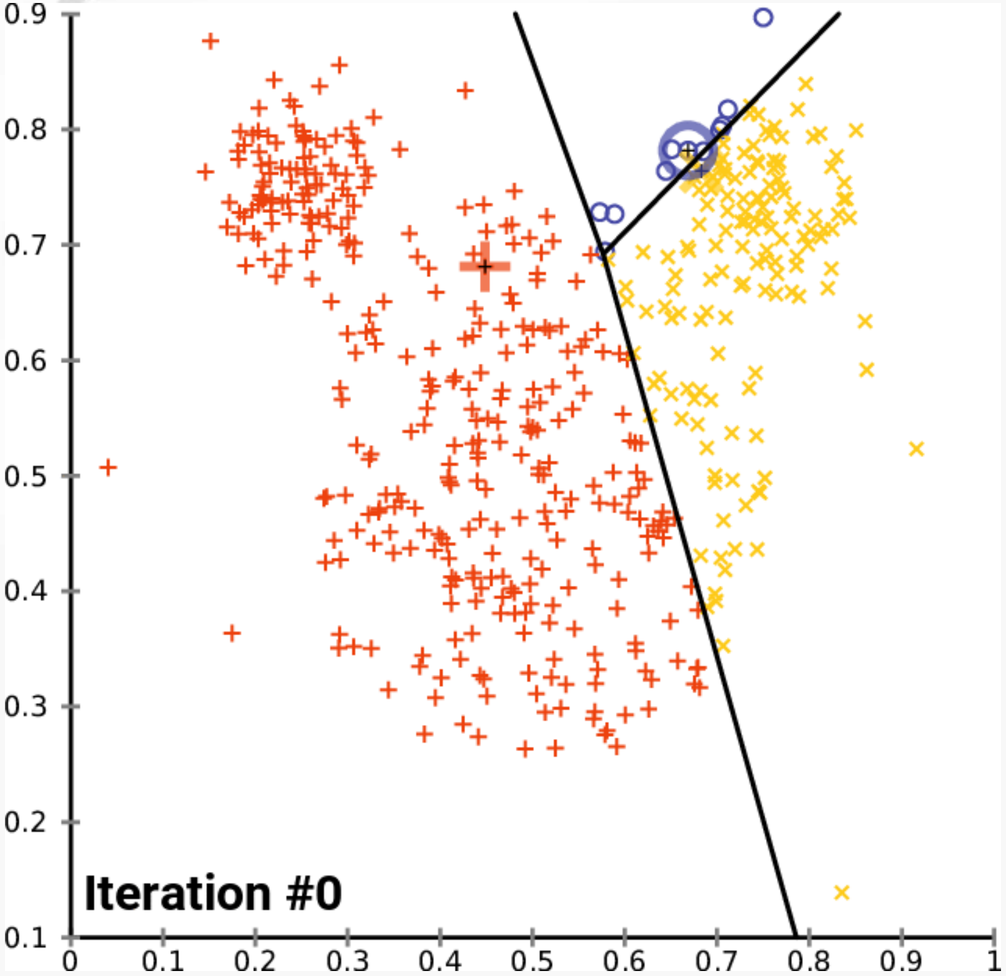
运行Alink脚本

Outline

- How to achieve high performance?
  - K-Means: from demo to state-of-the-art
  - Linear Models, Trees, GBDT, …
  - Utilities

K-Means Algorithm

Apache Flink



- Steps:
  1. Initialize K centroids;
  2. Assign each point to the nearest centroid, then have K clusters
  3. Calculate the centroid of each clusters;
  4. Repeat 2-3 until the centroids no longer change (converge).

Code from Flink Example:

```
class Point(var x: Double, var y: Double) extends Serializable {
…
class Centroid(var id: Int, x: Double, y: Double) extends Point(x, y) {
…
val points: DataSet[Point] = getPointDataSet(env)
val centroids: DataSet[Centroid] = getCentroidDataSet(env)

val finalCentroids = centroids.iterate(numIterations) { currentCentroids =>
  val newCentroids = points
    .map(new SelectNearestCenter).withBroadcastSet(currentCentroids, "centroids")
    .map { x => (x._1, x._2, 1L) }
    .groupBy(0)
    .reduce { (p1, p2) => (p1._1, p1._2.add(p2._2), p1._3 + p2._3) }
    .map { x => new Centroid(x._1, x._2.div(x._3)) }
  newCentroids
}
```

Only support 2-D double points

flink-examples/flink-examples-batch/src/main/scala/org/apache/flink/examples/scala/clustering/KMeans.scala

Data Representation

Apache Flink

Point

Row

- Defects of Point
  - Unable to process higher dimensions of features than 2
  - Types are fixed

- Benefits of Row
  - Support variable sizes, meeting the requirements of real-world clustering problems with tens, hundreds, or thousands of features
  - Support flexible feature types: boolean, integer, string, etc.

Adult dataset:
Row.of(39, “State-gov”, 77516.0, “Bachelors”, 13.0, “Never-married”, …)

Data Representation

Apache Flink

Dataset<Point>

↓

Dataset<Row>

↓

Table

- Limitation of Dataset<Row>
  - ML algorithm can't work without knowing the schema from data sources
  - Can't be store into databases, or be piped to other algorithms
- Benefits of Table
  - Schema is stored with data values
  - Connect to Table API, SQL and ML operations without additional structures
  - Convenient for users to run ML algorithms

Distance Type & Initial Centroids

Apache Flink

Distance Type
Euclidean
Cosine
CityBlock(Manhattan)
Haversine(GreatCircle)
Hamming

- K-means : non-convex optimization
- Random initial centroids + Lloyd(EM) algorithm could converge to very bad local optima.
- Solution:
  - K-means++ [1]
  - Parallel K-means++ [2]
- K-means++ algorithm can provide better convergence speed too.

[1] "k-means++: the advantages of careful seeding", Arthur et al., SODA 2007
[2] "Scalable K-means++", Bahmani et al., VLDB 2012

Slow Training Speed

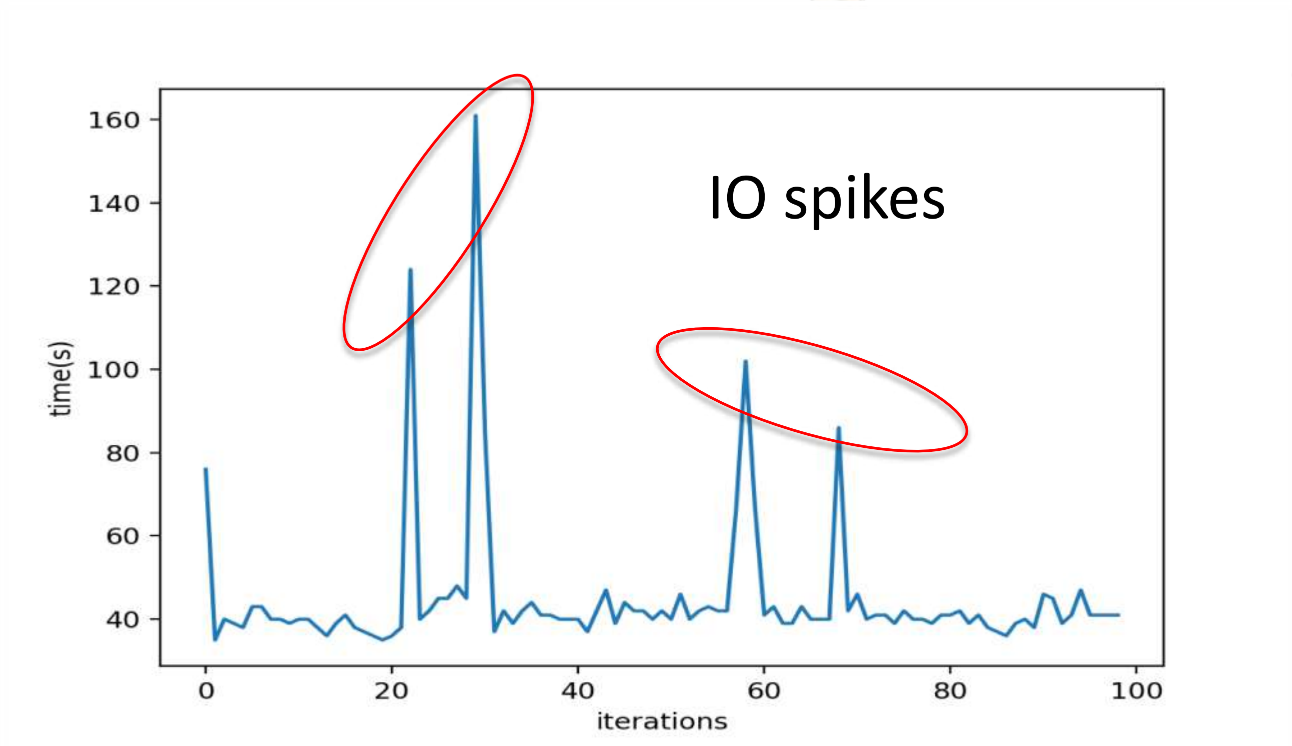
Apache Flink

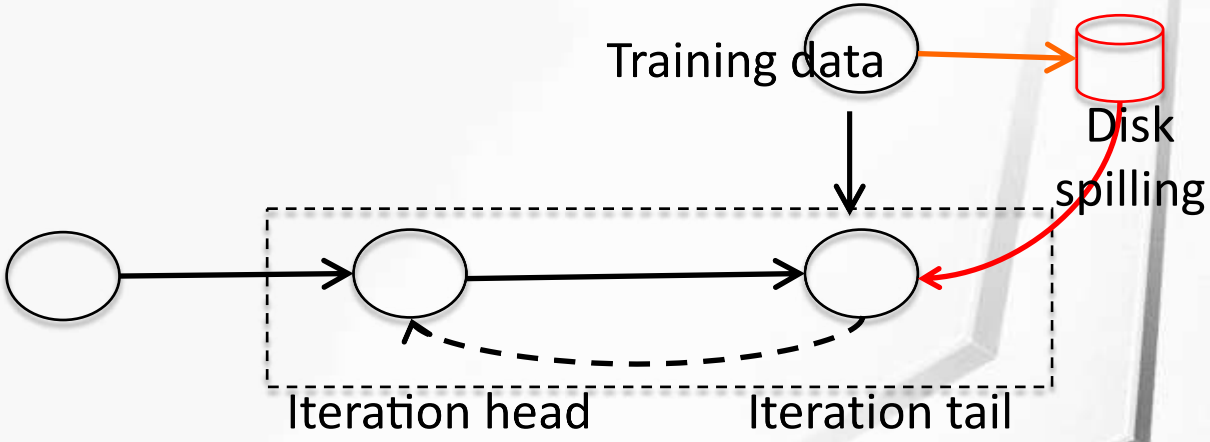
- Refine KMeans implementation
  - Calculation cost
  - Communication cost
  - Minimal (1) synchronization per iteration step
- Practice:
  - Still 30% slower than Spark ML algorithm

Mem-Level Cache

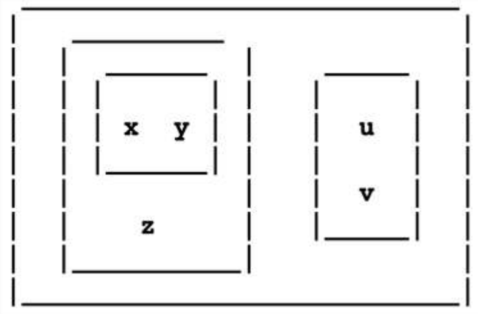
- Observation from experiment details:
  - The slow-down overlaps with temporary IO spike of some worker machines, influenced by other jobs
  - Flink spills dataset to disk, then loads them at the beginning of each iterations

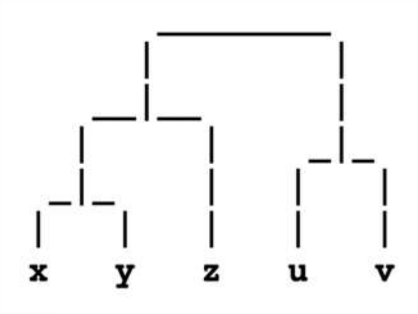
- Solution: mem-level cache



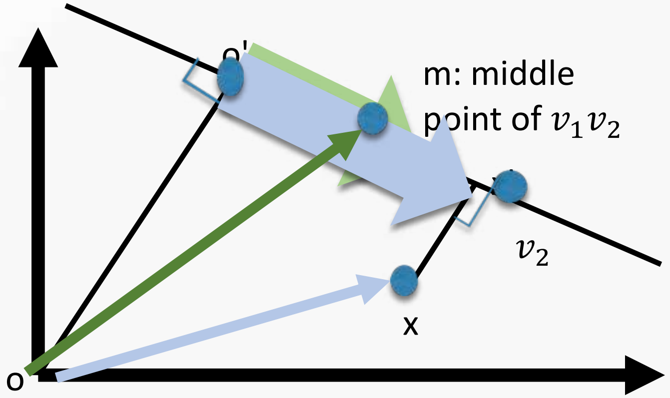


Improving Bisecting K-Means





- What is bisecting kmeans?
  - A kind of hierarchical clustering algorithm using "top-down" approach
  - Faster than K-Means and other bottom-up hierarchical clustering algorithms
- How do we outperforms SparkML's impl.?
  - A clever formula to select the nearest neighbor out of two points



- Speedup over SparkML's impl.
  - 1.8x

Normal implementation:

cluster $= |x - v_1|^2 > |x - v_2|^2 \; ? \; v_2 : v_1$

2x vector subtraction, 2x dot production

Our implementation:

$m = (v_1 + v_2) * 0.5$ pre-computed

$v = v_2 - v_1$ pre-computed

$p = m \cdot v$ pre-computed

$q = x \cdot v$ for each x

cluster = $q > p \; ? \; v_2 : v_1$

1x dot production

Outline

Apache Flink

- How to achieve high performance?
  - K-Means: from demo to state-of-the-art
  - Linear Models, Trees, GBDT, …
  - Utilities

The Structure of Linear System

Apache Flink

LogisticReg | LinearSvm | Perceptron

Binary Classification

Softmax

Multiple Classification

LassoReg | RidgeReg | LinearReg | SurvivalReg

Regression

Loss Function | Regularization

Unified representation: trainer, predictor, model

GD | SGD | LBFGS | OWLQN | NEWTON

Parallel Optimizer

GD | SGD | LBFGS | OWLQN | NEWTON

Local Optimizer

Version of LBFGS Optimizer

Apache Flink

v1 — Initial Version — 0.1x

First version without any optimization.

v2 — Basic opt strategy — 0.3x

1. Reduce broadcast data; 2. Improve data locality; 3. Avoid data serialization.

v3 — MemCachedDataSet — 0.7x

Avoid disk reading by storing train data in static memory at the first iteration.

v4 — Static Model — 1.0x

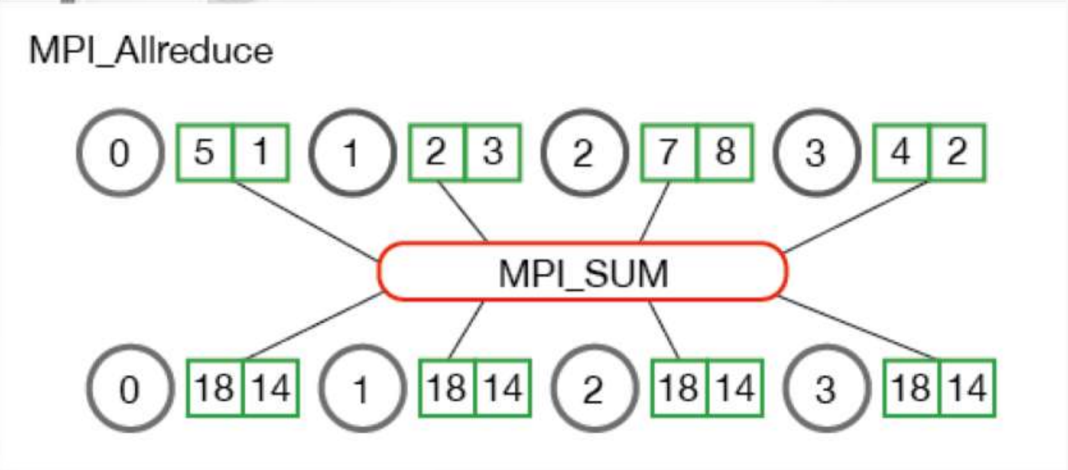
Store model in static memory, reduce model transmission during iterations.

v5 — AllReduce — 1.2x

Use AllReduce instead of reduce and broadcast to improve data communication.

Spark ML performance

AllReduce




```
// set number of bulk iterations for SGD linear Regression
IterativeDataSet<Params> loop = parameters.iterate(iterations);

DataSet<Params> newParameters = data
    // compute a single step using every sample
    .map(new SubUpdate())
    .withBroadcastSet(loop, "parameters")
    // sum up all the steps
    .reduce(new UpdateAccumulator())
    // average the steps and update all parameters
    .map(new Update());

// feed new parameters back into next iteration
DataSet<Params> result = loop.closeWith(newParameters);
```

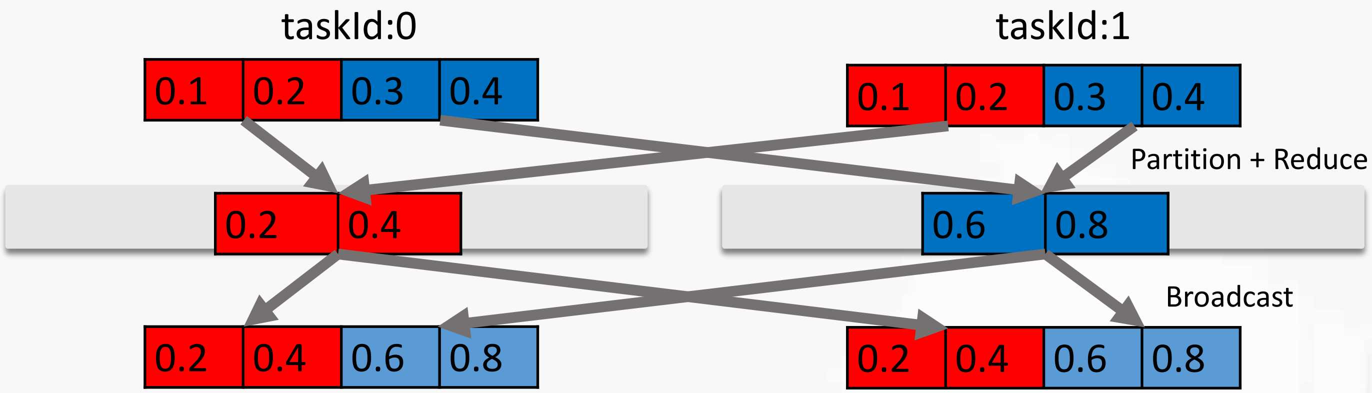


Illustration of AllReduce

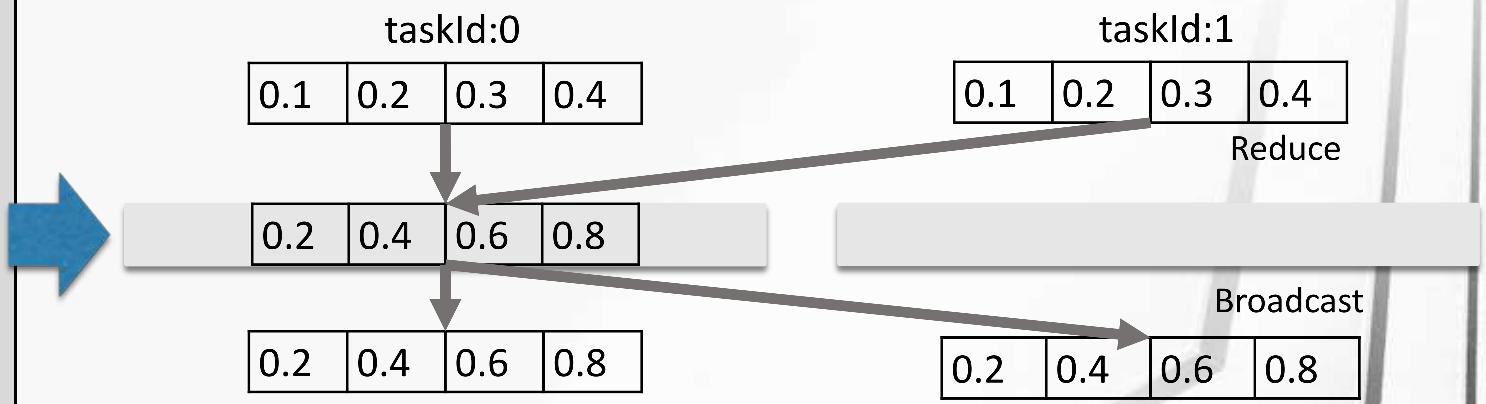
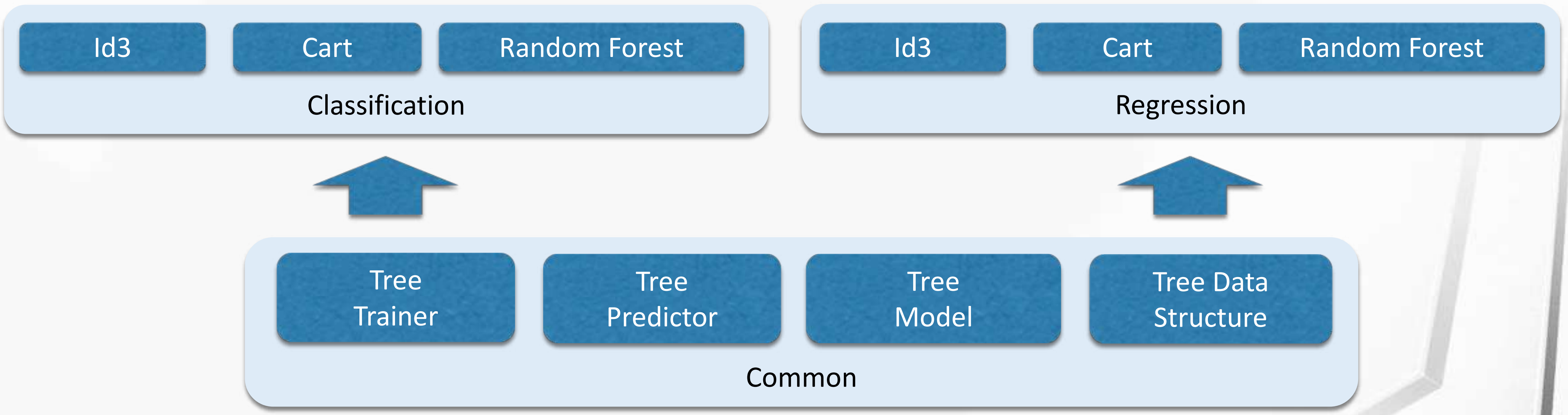


Illustration of Reduce + Broadcast

Decision Tree and Random Forest

- Use AllReduce Function
- Use pre allocation primitive type array to avoid gc and speed up
- Use MemCache to improve performance
- Add histogram subtraction to reduce the traffic [1]



[1] https://lightgbm.readthedocs.io/en/latest/

Outline

- How to achieve high performance?
  - K-Means: from demo to state-of-the-art
  - Linear Models, Trees, GBDT, …
  - Utilities

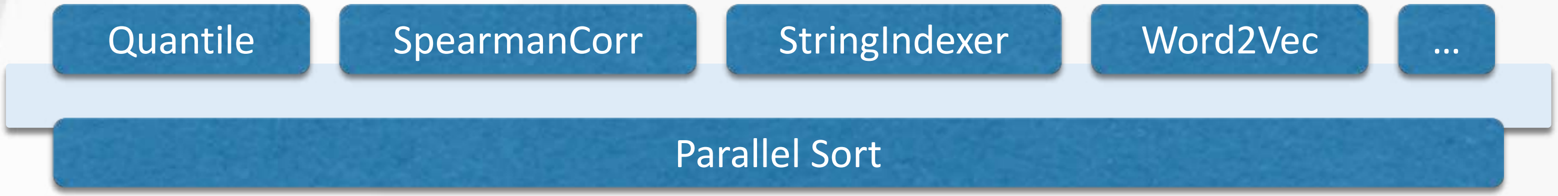
Utilities

- Optimizer
- MemCachedDataset
- AllReduce
- Linear algebra (base on blas/breeze)
- Parallel sort
- Statistics

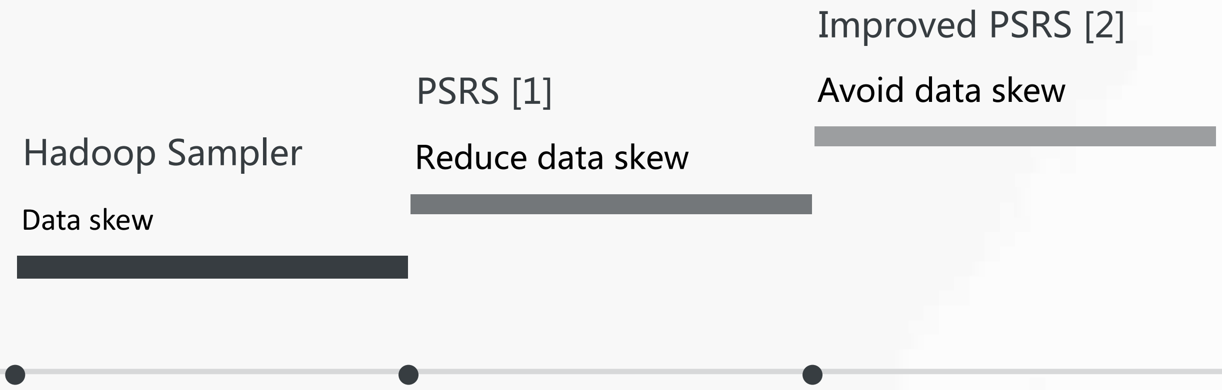
Apache Flink

Parallel Sort

- Base component of data processing



- Parallel Sorting by Regular Sampling[1]
- Use <TaskId, Object> as Object to avoid data skew[2]

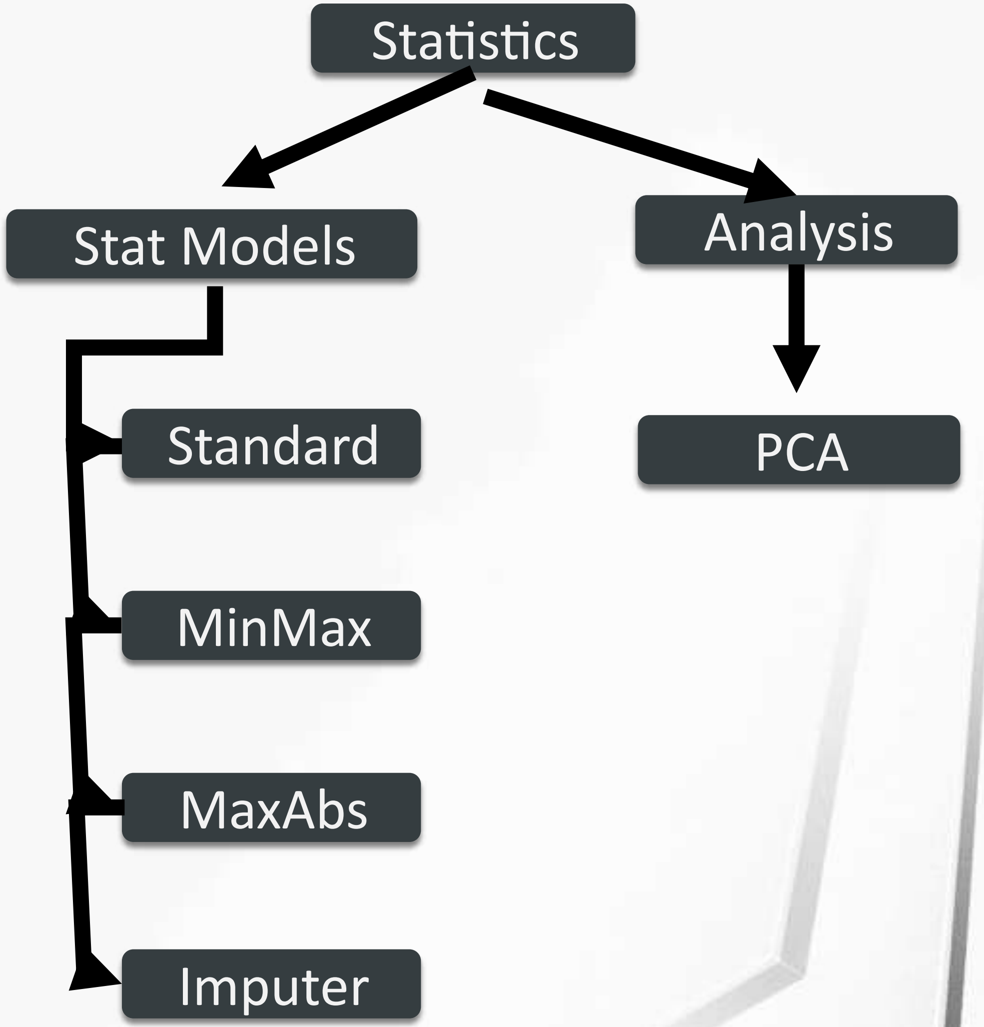


[1] http://csweb.cs.wfu.edu/bigiron/LittleFE-PSRS/build/h atml/PSRSalgorithm.html
[2] Yang, X. Refactoring statistics of big data (1st ed.), pp. 25-29, 2014 (Chong Gou Da Shu Ju Tong, In Chinese).

Statistics

We have statistics based on vector.

- Basic statistic: count, min, max, mean, variance, standard deviation, norm1, norm2.
- Covariance and correlation coefficient.
- Analysis: PCA



Statistics is not only an algorithm, but also a basic library for other algorithms, users can easily do secondary development based on statistics.

Current Flink ML Library

Apache Flink

- Supervised Learning
  - SVM
  - Multiple linear regression
  - Optimization Framework
- Unsupervised Learning
  - k-Nearest neighbors join
- Data Preprocessing
  - Polynomial Features
  - Standard Scaler
  - MinMax Scaler
- Recommendation
  - Alternating Least Squares (ALS)
- Outlier Selection
  - Stochastic Outlier Selection (SOS)
- Utilities
  - Distance Metrics
  - Cross Validation

开源

- 原先的FlinkML会下线
- 社区已经接收了新版FlinkML的接口设计，正提交算法代码
  - 准备开源的算法，会覆盖SparkML的全部算法功能，并且在性能上与SparkML持平或超过
  - 除了算法实现，还包括优化的算法框架及工具库，帮助开发者更容易基于Flink开发算法
  - 与社区的贡献者合作，一起建设强大的FlinkML
  - 后续还会分批开源更多的算法

Apache Flink

THANKS

【2群】Apache Flink C...

扫一扫群二维码，立刻加入该群。