



# 时间序列数据挖掘

汇报人：刘世喆



- [引言](#)
- [时间序列相似性度量](#)
- [时间序列查找](#)
- [时间序列预测](#)
- [时间序列分类](#)
- [时间序列聚类](#)
- [时间序列异常检测](#)
- [时间序列数据库](#)
- [时间序列智能运维](#)



---

# 引 言

---

誠 樸 雄 偉  
勵 學 敦 行

时间序列数据是基于时间的一系列数据点的集合，在有时间的坐标中将这数据点连成线，从时间维度往前看可以做成多维度报表，揭示其趋势性、规律性、异常性；往未来看可以做大数据分析、机器学习、实现预测和预警。

多元时间序列数据  $TSD = \{e_1, \dots, e_N\}$  由  $N$  个元素组成，元素  $e_i = (t_i, a_1, \dots, a_d)$  包含一个时间戳和  $d$  维数值型特征，且数据是按照时间顺序排列的，即如果  $i < j$ ，则  $t_i < t_j$ 。



图1：2022年8月19日茅台股票价格

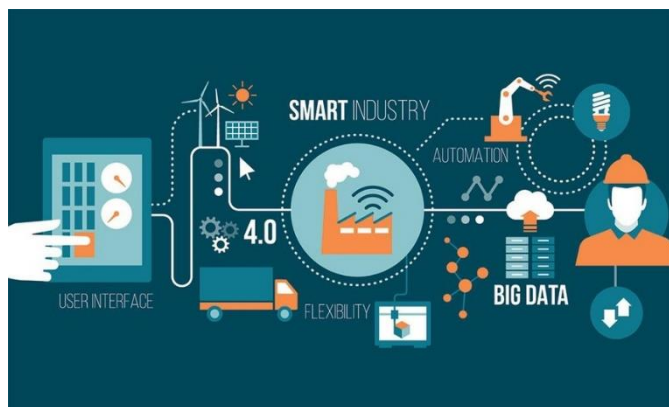
# 时间序列数据来源



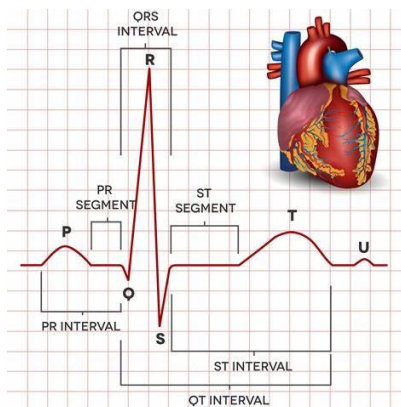
南京大學

NANJING UNIVERSITY

- ❑ IOT传感器：如温度传感器、压力传感器、湿度传感器等
- ❑ 医疗设备：如心电图（ECG）和脑电图（EEG）
- ❑ 金融市场数据：股票价格，股票交易数据
- ❑ .....



(a)工业物联网各传感器



(b)心电图



(c)金融市场数据

图2：时间序列数据来源

## □ 特点

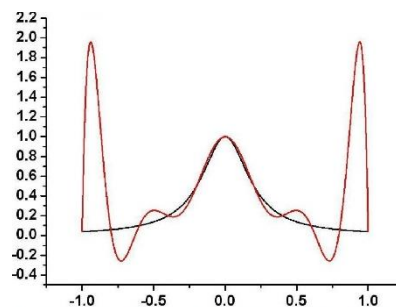
- 数据量大→从中挖掘有用信息困难
- 数据产生速度快→要求处理速度快
- 数据越新价值越大→更关心最新数据

## □ 应用

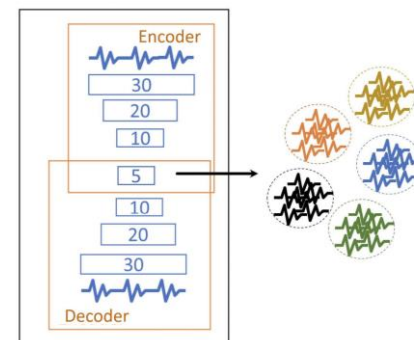
- 金融：发现股票时间序列可能存在的规律、发现市场风险出现的规律
- 生物和医学领域：检测大脑异常放电活动、检测功能相似的基因
- 交通、环境：检测道路车流量、地震波异常检测

## 数据自适应

- 分段多项式插值(PPI)
- 分段聚合近似(PAA)
- 符号聚合近似(SAX)



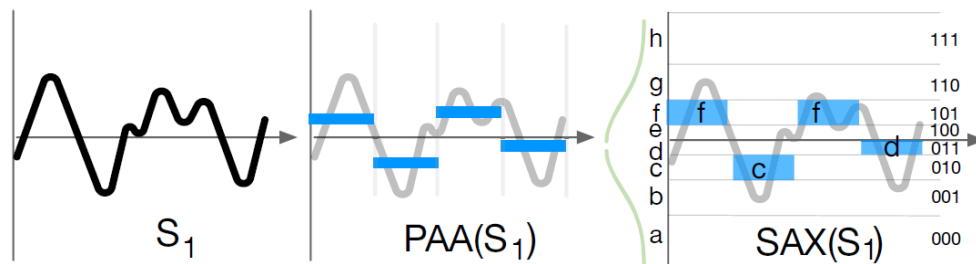
(a)分段多项式插值



(b)深度学习模型

## 非数据自适应

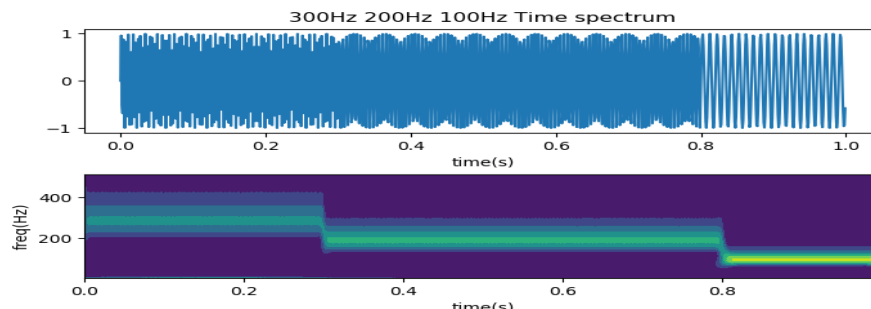
- 离散傅里叶变换(DFT)
- 离散小波变换(DWT)



(c)分段聚合近似和符号聚合近似

## 基于模型

- 隐马尔可夫模型(HMM)
- 自回归移动平均(ARMA)
- 深度学习模型(DLM)



(d)离散小波变换

图3: 时间序列数据部分表示方法



---

# 时间序列相似性度量

---

誠樸雄偉  
勵學敦行



如何区分或匹配任意两个时间序列？如何标准化两个时间序列之间的认知距离？

➤ 时间序列相似性度量，选取一个距离函数来衡量时间序列 $X$ 和 $Y$ 的相似性

相似性度量对于时序数据分类、聚类、异常检测、模式挖掘等应用至关重要！

Distance functions are fundamental to the effective design of data mining algorithms, because a poor choice in this respect may be very detrimental to the quality of the results.

**Aggarwal, Data Mining, Chapter 3.**

- 闵可夫斯基距离(Minkowski distance)
- 余弦相似度(Cosine Similarity)
- 马氏距离(Mahalanobis distance)
- 动态时间规整 (Dynamic Time Warping, DTW)

表1: 时间序列相似性度量方法对比

度量方法	计算开销	优点	缺点
闵可夫斯基距离	低	实现简单, 易拓展	不能辨别形状, 易受异常值影响
余弦相似性	低	能体现方向上的相对差异	不能衡量每个维数值的差异
马氏距离	高	能排除变量之间相关性的干扰	计算不稳定(受协方差矩阵不稳定影响)
动态时间规整	高	不要求数据等频等长	计算复杂度大

# 相似性度量方法计算公式



假设两个一元时间序列数据为 $X$ 和 $Y$ ，其中

$$X = \{(t_1, x_1), \dots, (t_n, x_n)\}$$

$$Y = \{(t_1, y_1), \dots, (t_m, y_m)\}$$

于是两者闵可夫斯基距离： $Mk(X, Y) = (\sum_{i=1}^m |x_i - y_i|^p)^{\frac{1}{p}}$ ，当 $p = 2$ 时，为欧式距离。

$$\text{余弦相似性: } \cos \langle X, Y \rangle = \frac{\sum_{i=1}^m x_i \cdot y_i}{\|X\| \cdot \|Y\|}$$

马氏距离： $Ma(X, Y) = \sqrt{(X - Y)^T \Sigma^{-1} (X - Y)}$ ，其中 $\Sigma$ 是多维随机变量的协方差矩阵

$X$ 和 $Y$ 等频等长!

# 动态时间规整(DTW)方法介绍

DTW是一种将时间规整和距离测度相结合的一种非线性规整技术。主要思想是把未知量均匀地伸长或者缩短，直到与参考模式的长度一致，在这一过程中，未知量的时间轴要不均匀地扭曲或弯折，以使其特征与参考模式特征对正。

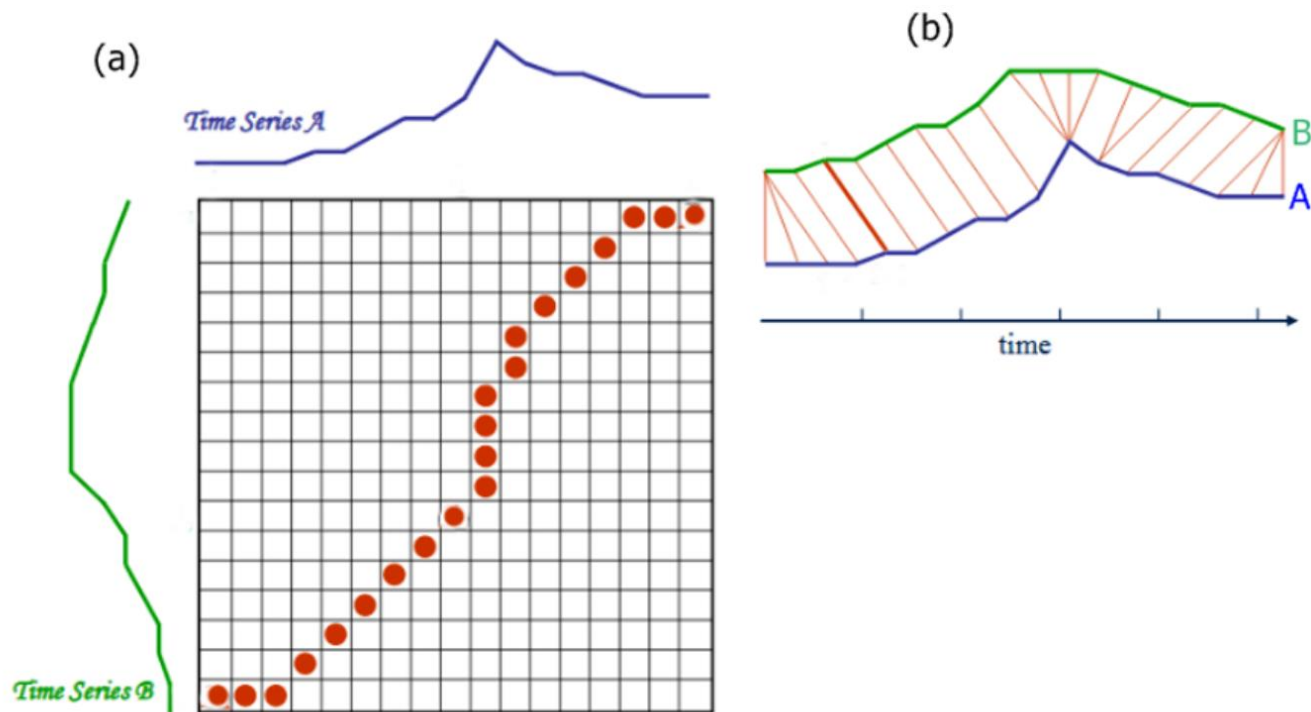


图4: 动态时间规整示意图

假设两个时间序列数据 $X = \{(t_1, x_1), \dots, (t_n, x_n)\}$ 和 $Y = \{(t_1, y_1), \dots, (t_m, y_m)\}$ .

构造一个 $n \times m$ 矩阵 $A$ ,  $a_{ij}$ 记录两个点 $x_i$ 和 $y_j$ 之间的欧式距离, 即

$$a_{ij} = d(x_i, y_j) = |x_i - y_j|$$

$X$ 和 $Y$ 的距离由一条弯折的路径 $W$ 表示,  $W$ 由若干个彼此相连的矩阵单元构成, 这条路径描述了 $X$ 和 $Y$ 之间的一种映射。设第 $k$ 个单元定义为 $w_k = (i, j)_k$ , 则

$$W = w_1, \dots, w_k, \dots, w_K$$

其中 $\max(n, m) \leq K \leq n + m + 1$ .

这条弯折的路径满足如下的条件:

- 1) 边界条件:  $w_1 = (1, 1)$ , 且 $w_k = (n, m)$
- 2) 连续性: 设 $w_k = (a, b)$ ,  $w_{k-1} = (a', b')$ , 那么 $a - a' \leq 1$ ,  $b - b' \leq 1$
- 3) 单调性: 设 $w_k = (a, b)$ ,  $w_{k-1} = (a', b')$ , 那么 $a - a' \geq 0$ ,  $b - b' \geq 0$

在满足上述条件的多条路径中，最短的，花费最少的一条路径是：

$$DTW(X, Y) = \min \frac{1}{K} \sqrt{\sum_{k=1}^K w_k}$$

DTW距离的计算过程是一个动态规划过程，先用欧式距离初始化矩阵，然后使用如下递推公式进行求解：

$$r(i, j) = d(i, j) + \min\{r(i-1, j-1), r(i-1, j), r(i, j-1)\}$$



---

# 时间序列查找

---

誠樸雄偉  
勵學敦行

给定一个时间序列数据片段  $Q = \{(t_1, v_1), \dots, (t_l, v_l)\}$ ，如何快速查询在整个时间序列数据中与  $Q$  最相似的  $k$  个数据段（假设数据产生速率是恒定的）？

表2：时间序列查找方法对比

方法	概述	是否可行	理由
暴力搜索(Brute-Force Search)	顺序遍历整个数据集查找相似序列	否	时间序列数据量大，查找开销大
聚类(Clustering)	将相似的序列聚到一起以便查找	是	聚类可以将相似数据聚在一起，但需多次遍历整个数据集，构造开销大
摘要法 (Summarization)	使用降维技术(如PAA, SAX) 将时间序列数据转换为摘要	是	可近似估算数据序列之间的相似性，以便过滤掉大部分不相关时间序列
局部敏感哈希 (Locality-Sensitive Hashing, LSH)	相邻的高维数据投影到低维空间大概率仍相邻，不相邻的高维数据投影到低维空间大概率不相邻	是	高纬度数据映射到低纬度空间中，相邻关系仍能大概率保持下来，于是可以从低维空间中去查找相似数据，从而过滤掉大部分不相关的时间序列

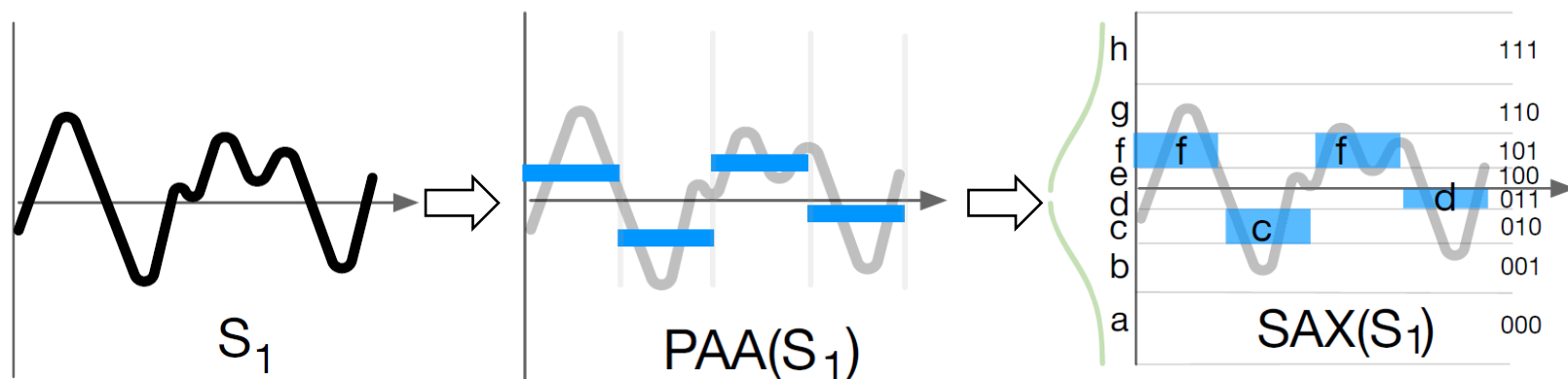


# 数据降维：符号聚合近似(SAX)技术

SAX算法步骤:

**Step 1:** 分段聚合近似(PAA)。将时间序列数据划分为 $w$ 个大小相等的段，并为每个段计算其平均值;

**Step 2:** 离散化值空间并为每段分配一个符号。为不同范围的值划分空间区域(区域划分遵循正态分布), 以便让原始数据序列值在各个区域之间的分布概率大致相等(假设已经将数据归一化), 将二进制串(符号)分配给每个区域。根据这一段序列数据的PAA 值所在区域的符号作为这段的数据摘要。



$$SAX(S_1) = (f, c, f, d) = (101, 010, 101, 011)$$

图5: PAA和SAX摘要

# SAX相似性度量

给定两个时间序列数据 $T$ 和 $S$ (等频等长), 其欧式距离为:  $D(T, S) = \sqrt{\sum_{i=1}^n (T_i - S_i)^2}$

给定两个时间序列数据 $T$ 和 $S$ (等频等长)的SAX表示为 $SAX_T$ 和 $SAX_S$ , 其近似欧式距离

$$AD(SAX_T, SAX_S) = \sqrt{\frac{n}{w}} \sqrt{\sum_{i=1}^w (dist(T_{syb_i}, S_{syb_i}))^2}$$

其中,  $T_{syb_i}$ 表示时间序列 $T$ 第 $i$ 个段对应的SAX符号,  $S_{syb_i}$ 表示时间序列 $S$ 第 $i$ 个段对应的SAX符号,  $dist$ 函数用于计算两个SAX符号的距离(可以通过查表确定)。

表3: 断点表( $a$ 表示使用符号数量,  $\beta_i$ 表示断点位置)

$\beta_i \backslash a$	2	3	4	5	6	7	8
$\beta_1$	0.00	-0.43	-0.67	-0.84	-0.97	-1.07	-1.15
$\beta_2$		0.43	0.00	-0.25	-0.43	-0.57	-0.67
$\beta_3$			0.67	0.25	0.00	-0.18	-0.32
$\beta_4$				0.84	0.43	0.18	0.00
$\beta_5$					0.97	0.57	0.32
$\beta_6$						1.07	0.67
$\beta_7$							1.15

表4: SAX距离查询表(使用符号数量为8)

	000	001	010	011	100	101	110	111
000	0	0	0.48	0.83	1.15	1.47	1.82	2.30
001	0	0	0	0.35	0.67	0.99	1.34	1.82
010	0.48	0	0	0	0.32	0.64	0.99	1.47
011	0.83	0.35	0	0	0	0.32	0.67	1.15
100	1.15	0.67	0.32	0	0	0	0.35	0.83
101	1.47	0.99	0.64	0.32	0	0	0	0.48
110	1.82	1.34	0.99	0.67	0.35	0	0	0
111	2.30	1.82	1.47	1.15	0.83	0.48	0	0

# 符号近似聚合索引技术(iSAX)

iSAX是一个树形结构，由根节点、内部节点、和叶子节点组成。

根节点代表整个 SAX 空间。

内部节点对应于来自所有段的公共 SAX 前缀，存放在内存中，并缓冲部分叶节点数据。

叶子节点存放在磁盘中，存储时间序列数据SAX表示和其原始数据对应的磁盘位置。

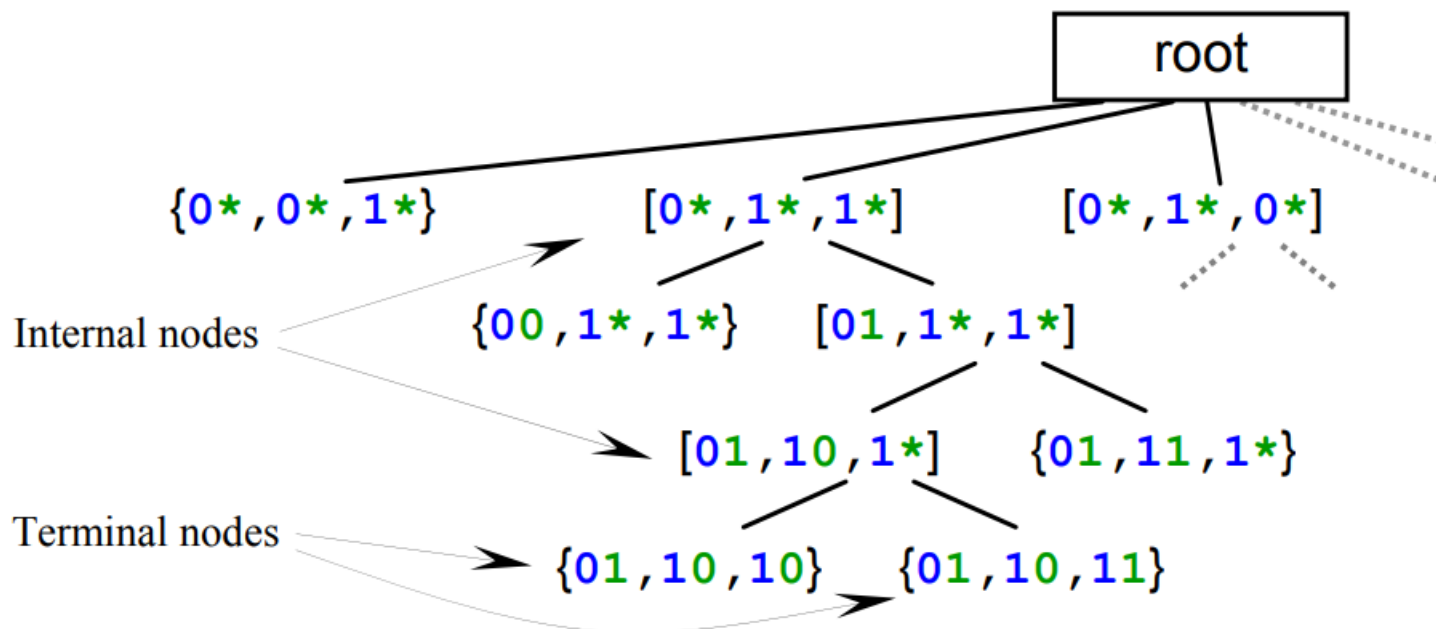


图6: 符号近似聚合索引示意图

当插入一个时间序列数据片段时，首先计算其SAX值，然后从根节点出发，找到对应的内部节点，将该数据插入到内部节点的缓存中。内部节点的缓存满了会将数据刷新到叶子节点中。当一个叶子节点填满时，需要将这个叶子节点拆分成两个新叶子节点去存储相关数据，并生成两个内部节点。

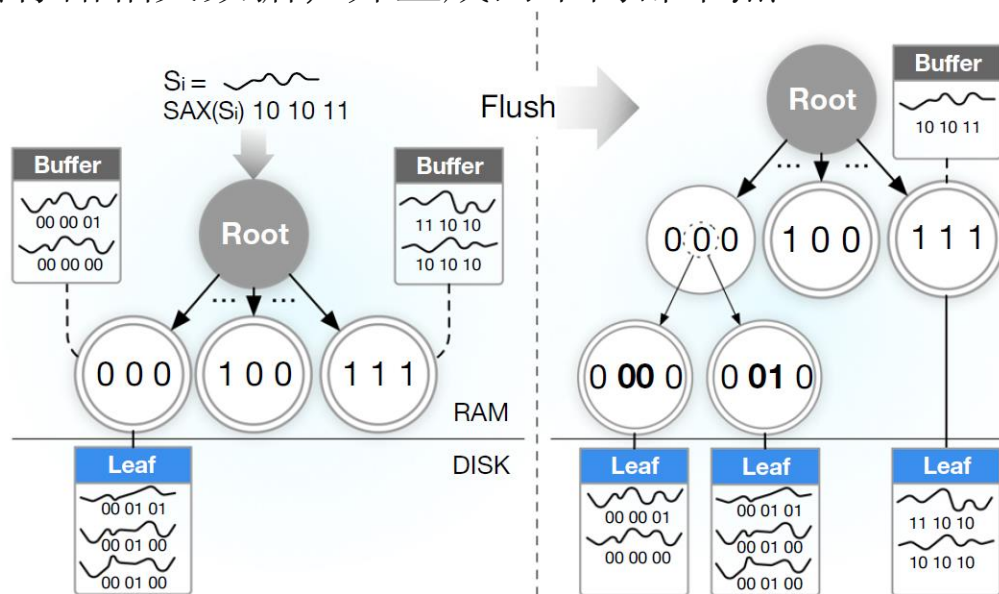


图7: 向索引结构插入 $S_i$ 过程示意图, 此时最左边和最右边内部节点缓存已满, 需要刷新到叶子节点中。刷新后, 内部节点  $[0^*, 0^*, 0^*]$  对应的叶子节点满, 导致需要分裂成两个新内部节点  $[0^*, 00, 0^*]$  和  $[0^*, 01, 0^*]$ 。新的叶子节点同样需要分配空间去存储新的数据。

**近似查找：**两个相似的时间序列通常SAX表示也相同。于是在索引中搜索与查询序列具有相同SAX表示的叶子节点来获得近似结果；如果在索引中不存在该SAX表示，则选择搜索过程中与查询时间序列数据SAX表示最相邻的叶子节点。

**精确查找：**使用广度搜索去遍历树节点，确定哪些叶子节点满足相似条件，然后与满足条件的每个叶子节点时间序列数据进行准确的相似度计算，以此确定与查询数据最相似的 $k$ 个数据。

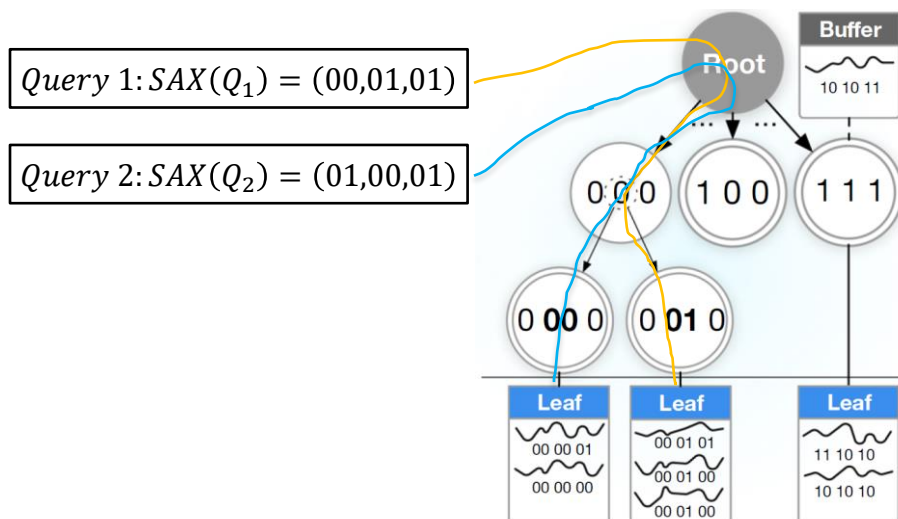


图8: iSAX查找例子

类似聚类的方法，是否存在相关技术将相似的数据存放在一起(不需要多轮数据训练)，当查询时，直接在相似的数据桶中去查找相关数据？

✓ 局部敏感哈希

基本思想：高维数据空间相邻的数据映射到低维数据空间后，大概率仍相邻；而原本不相邻的数据，在低维空间中也将有很大概率不相邻。

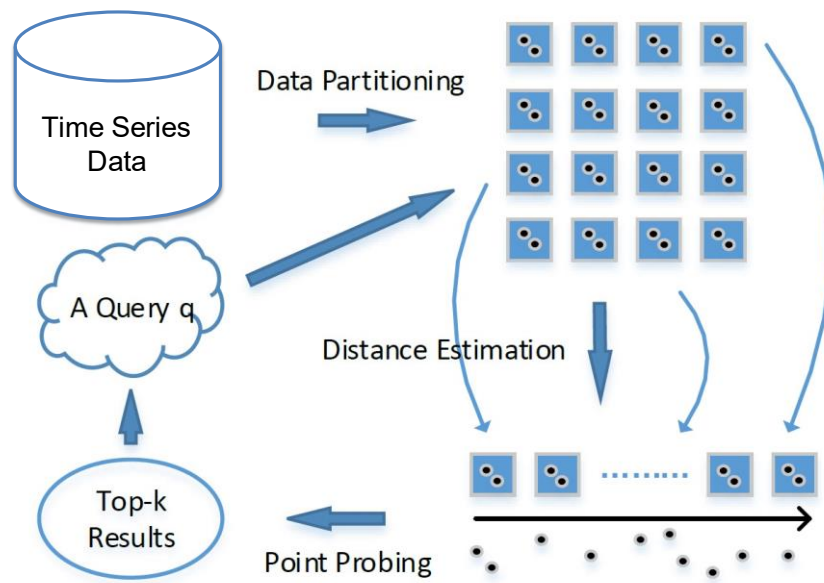


图9：局部敏感哈希通用架构

具有怎样特点的哈希函数才能够使得原本相邻的两个数据点经过哈希变换后会落入相同的桶内？这些哈希函数需要满足以下两个条件：

(1) 如果 $d(X, Y) \leq d_1$ ，则 $h(X) = h(Y)$ 的概率至少为 $p_1$ ；

(2) 如果 $d(X, Y) \geq d_2$ ，则 $h(X) = h(Y)$ 的概率至多为 $p_2$ ；

其中 $d(X, Y)$ 表示 $X$ 和 $Y$ 之间的距离， $d_1 < d_2$ ， $h(X)$ 和 $h(Y)$ 分别表示对 $X$ 和 $Y$ 进行哈希变换。满足以上两个条件的哈希函数称为 $(d_1, d_2, p_1, p_2) - sensitive$ 。而通过一个或多个 $(d_1, d_2, p_1, p_2) - sensitive$ 的哈希函数对原始数据集合进行哈希生成一个或多个哈希表的过程称为局部敏感哈希。

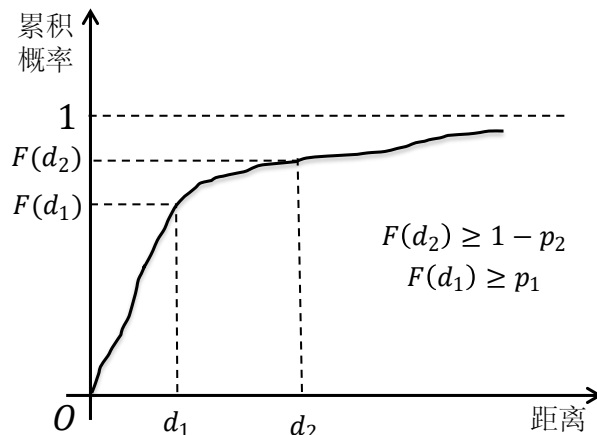


图10: 两个数据实例哈希值相等概率累积分布

使用局部敏感哈希进行对海量数据建立索引的过程如下：

Step 1: 选取满足 $(d_1, d_2, p_1, p_2) - sensitive$ 的局部敏感哈希函数；

Step 2: 根据相邻的数据被查找到的概率确定哈希表的个数 $L$ ，每个表内的哈希函数的个数 $K$ ，以及跟局部敏感哈希函数自身有关的参数；

Step 3: 将所有数据经过局部敏感哈希函数哈希到相应的桶内，构成了一个或多个哈希表。

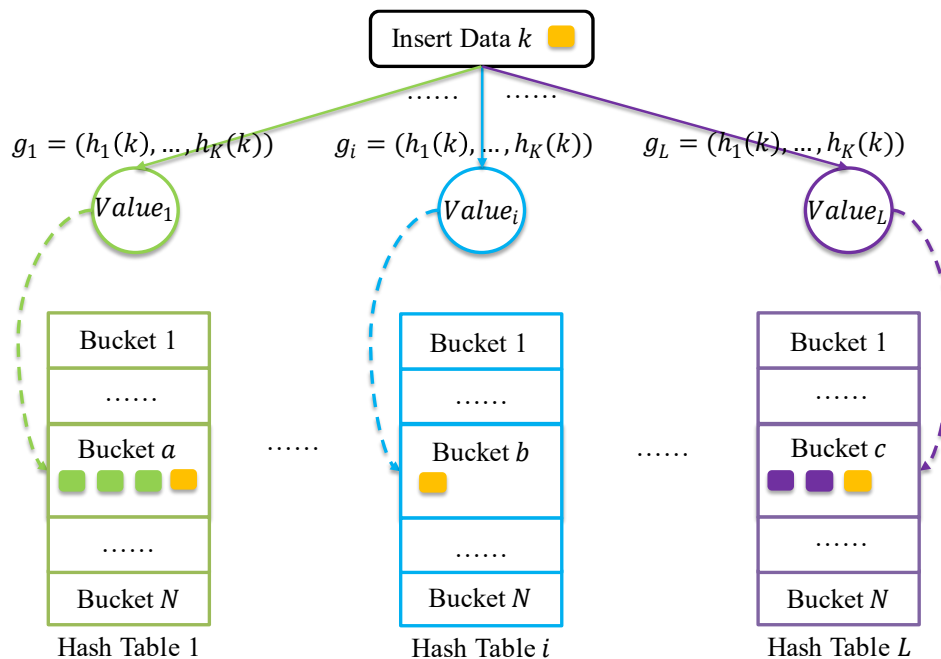


图11: 局部敏感哈希索引建立过程



# 局部敏感哈希k近邻查找



查找步骤:

Step 1: 将查询数据经过局部敏感哈希函数哈希得到相应的桶号;

Step 2: 将桶号中对应的数据取出;

Step 3: 计算查询数据与对应的桶数据之间的相似度或距离, 返回k近邻的数据。

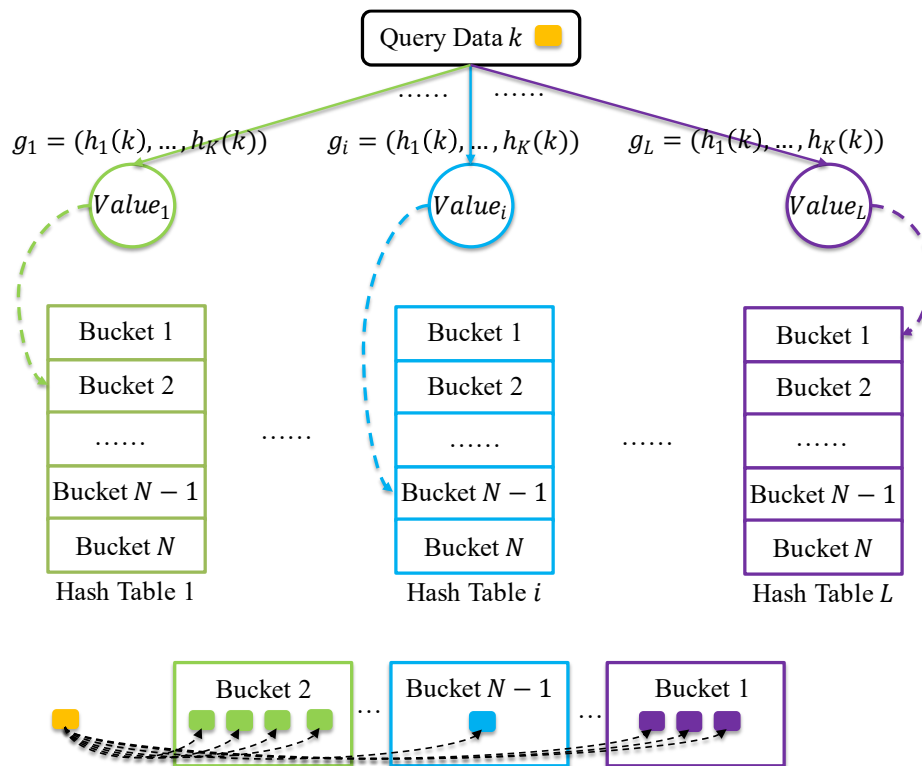


图12: 局部敏感哈希最近邻查找过程

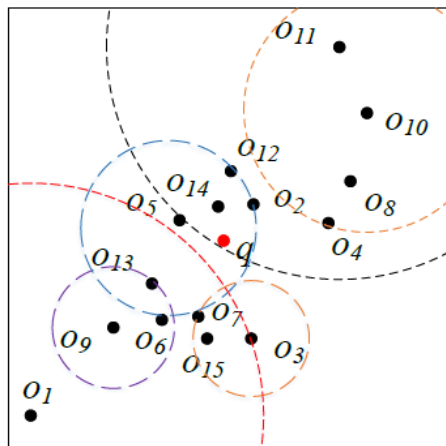
# 局部敏感哈希k近邻查找复杂度



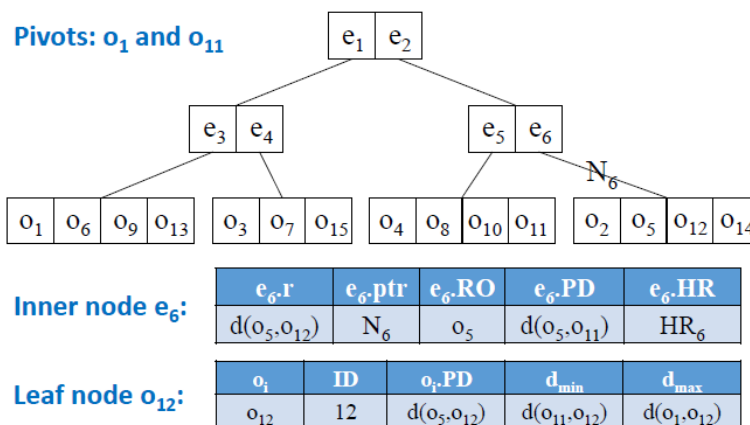
在线查找时间由两个部分组成：

- (1) 通过局部敏感哈希函数计算哈希值（桶号）的时间；
- (2) 将查询数据与桶内的数据进行相似性度量的时间。

LSH的查找时间至少是一个sublinear时间。因为可以通过对桶内的数据建立索引来加快匹配速度，此时第(2)部分的耗时就从 $O(N)$ 变成了 $O(\log N)$ 或 $O(1)$ (取决于采用的索引方法)。例如文献《PM-LSH: A fast and accurate LSH framework for high-dimensional approximate NN search》使用了PM-Tree加速匹配速度。



(a) Space Partitioning



(b) PM-tree

图13: 对每个桶建立PM-Tree加速匹配



---

# 时间序列预测

---

誠樸雄偉  
勵學敦行

时间序列预测：给定时间序列数据中依次顺序到达的 $n$ 个点 $\{(t_1, v_1), \dots, (t_n, v_n)\}$ ，预测接下来 $m$ 个点值 $\{(t_{n+1}, v_{n+1}), \dots, (t_{n+m}, v_{n+m})\}$ 。

对未来趋势的预测被应用于零售销售、经济指标、天气预报、股票市场等其他场景中。

对时间序列的预测方法可以分类三类，一类是基于自回归的预测模型，另一类是基于时间序列分解的预测模型，最后一类是基于机器学习的预测模型。

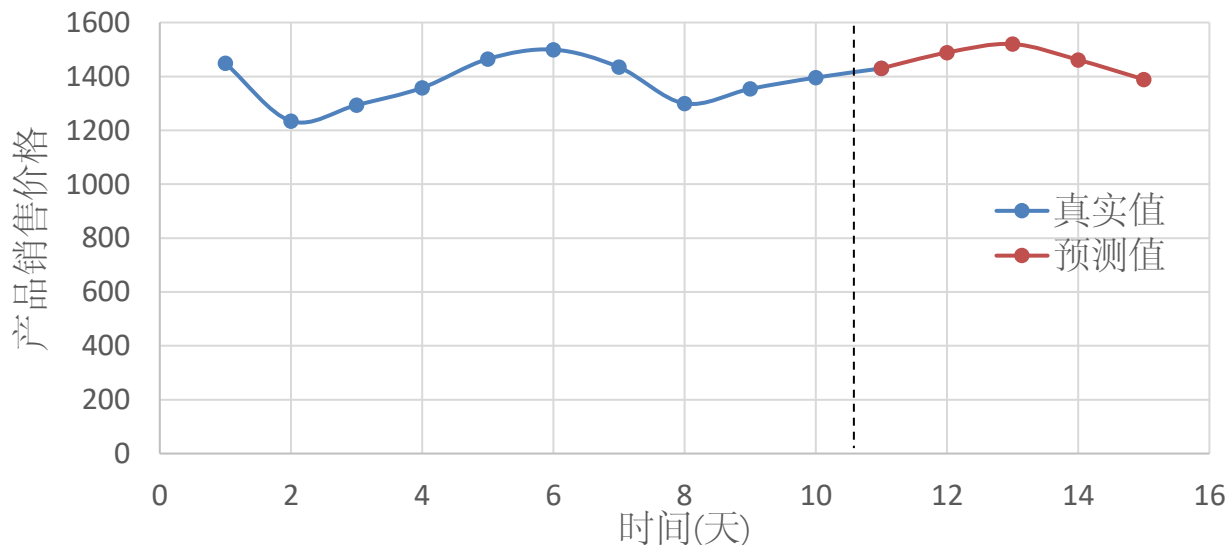
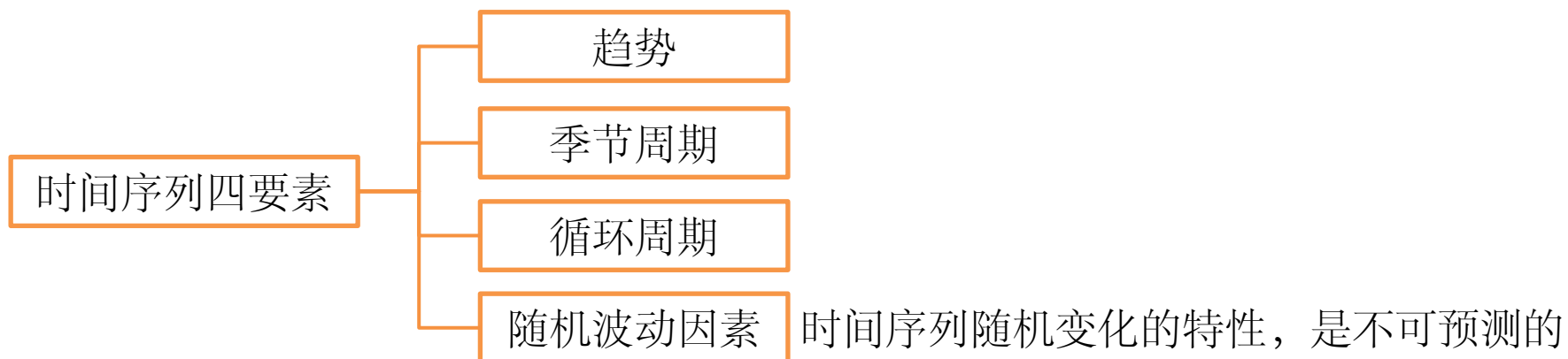
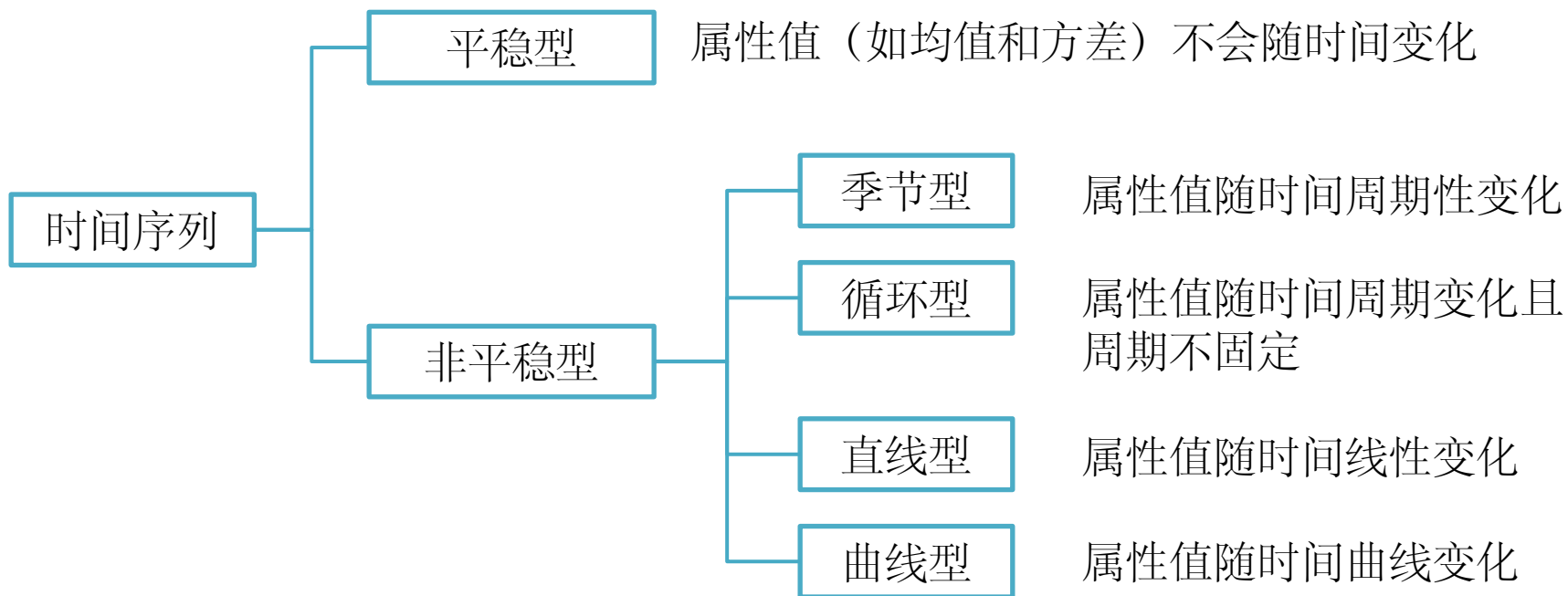


图14：时间序列预测示意图

- ❑ 磁盘、CPU的容量预测：当磁盘、CPU的容量占用较高时，可能会降低应用或者系统的运行性能，对磁盘、CPU的容量有效预测可以保证IT系统高效稳定运行
- ❑ 磁盘故障预警：海量数据的存储对应着更多的磁盘存储需求，而磁盘的频繁损坏导致的数据丢失将会给企业带来不可估量的损失，企业对于数据存储介质的稳定性和安全性的要求增高，让磁盘故障预警变得越来越重要
- ❑ 业务预测：如商业、金融这类领域，业务种类繁多，通过对各种类的业务量进行监控，可以更精准地制定相应的业务计划，并合理设置资源规划



- 自回归 (Auto Regressive, AR) 模型
- 移动平均 (Moving Average, MA) 模型
- 自回归移动平均 (Auto Regressive Moving Average, ARMA) 模型
- 自回归集成移动平均 (Auto Regressive Integrated Moving Average, ARIMA) 模型

基本思想：某些时间序列是依赖于时间 $t$ 的一组随机变量，构成该时间序列的单个序列值虽然具有不确定性，但整个序列的变化却有一定的规律性，可以用相应的数学模型近似描述。

一元时间序列包含一个单一变量，可以使用自相关性对其预测。自相关性表示序列中相邻位置时间戳之间的相关性。通常情况下，相邻位置时间戳上的行为属性值呈正相关。一个时间序列的自相关性由一个特俗的延迟值 $L$ 所定义。对于一个时间序列 $y_1, \dots, y_n$ , 定义延迟为 $L$ 的自相关性为 $y_t$ 和 $y_{t+L}$ 之间的皮尔森相关系数，即

$$\text{Autocorrelation}(L) = \frac{\text{Covariance}_t(y_t, y_{t+L})}{\text{Variance}_t(y_t)}$$

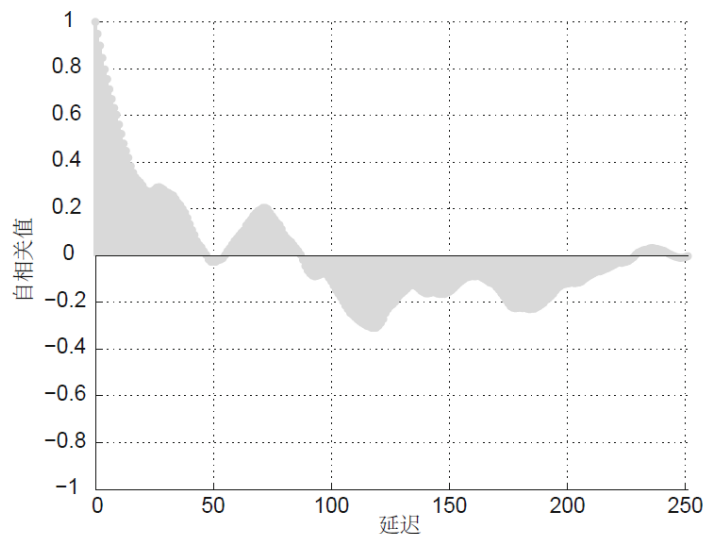


图15: IBM股票实例带有延迟的自相关性的变化



自回归模型定义时间 $t$ 处的值 $y_t$ 为前面紧邻长度为 $p$ 的窗口中值的线性组合：

$$y_t = \sum_{i=1}^p a_i \cdot y_{t-i} + c + \epsilon_t$$

使用前面的长度为 $p$ 的窗口模型称为 $AR(p)$ 模型。回归系数 $a_1, \dots, a_p, c$ 的值需要从训练数据中学习。 $p$ 值越大，引入的自相关性延迟越大。 $p$ 的取值对自回归模型准确性影响较大。需要选择一个合适的 $p$ 值，使得延迟 $L = p$ 时自相关作用比较小。

说明：

1. 系数 $a_1, \dots, a_p, c$ 可以用最小二乘法进行估计。
2. 只有当时间序列的关键属性值（如：均值、方差和自相关性）不会随时间显著变化时，AR模型才能有效地用于预测未来值。

MA模型根据过去的历史预测偏差对后续序列值进行预测。MA模型的定义如下：

$$y_t = \sum_{i=1}^q b_i \cdot \epsilon_{t-i} + c + \epsilon_t$$

该模型也称 $MA(q)$ 。参数 $c$ 是时间序列的均值； $b_1, \dots, b_q$ 的值是需要从数据中学习的数据； $\epsilon_t$ 是误差项，其值从预测模型中获取。

说明：

- 1.用迭代非线性拟合求解模型参数
- 2.仅使用历史震荡进行时间序列预测是比较少见的。MA模型使用的最佳场景：时间戳上的行为属性值依赖于时间序列的历史震荡，而不是实际的序列值。

无论是AR还是MA模型，都不能单独地捕捉到预测所需的所有相关性。将AR和MA模型相结合，会使得模型有更好的健壮性。ARMA模型将自回归项 $p$ 和移动平均项 $q$ 结合起来，定义如下：

$$y_t = \sum_{i=1}^p a_i \cdot y_{t-i} + \sum_{i=1}^q b_i \cdot \epsilon_{t-i} + c + \epsilon_t$$

该模型也称ARMA( $p, q$ )模型。

ARMA模型关键问题是参数 $p$ 和 $q$ 的选择。 $p$ 和 $q$ 值设置得太小，则模型不能很好拟合数据； $p$ 和 $q$ 值设置得太大，那么则可能过度拟合数据。

ARIMA模型也称为Box-Jenkins模型，是广泛应用于时间序列分析的常见模型，它可以用来处理包含季节趋势的时间序列。

ARIMA由三个参数组成，即自回归阶数 $p$ 、差分阶数 $d$ 和移动平均阶数 $q$ ，通常模型被写作 $ARIMA(p, d, q)$ 。

ARIMA模型处理步骤：

Step 1: 对数据求差分直到它是平稳的；

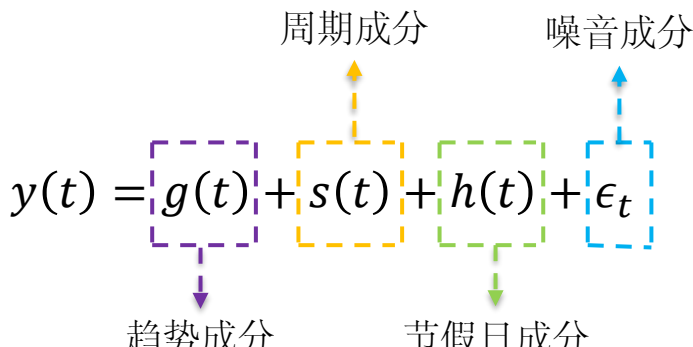
Step 2: 选定一个特定的模型拟合所分析的时间序列数据；

Step 3: 用时间序列的数据估计模型的参数并进行检验，以判定该模型是否恰当，如果不恰当，则返回Step 2，重新选定模型。

Prophet模型是基于时间序列分解的预测模型，由Facebook(现改名为Meta)公司于2017年提出，最初应用于其公司内部的流量预测，在开源后由于模型上手容易和泛用性强，得到了业内更为广泛的应用。

Prophet模型利用了时间序列分解的思想，将时间序列的趋势部分、周期部分以及节假日部分，分别建立关于时间轴的回归模型，并且通过贝叶斯的框架优化其回归系数。

Prophet模型假设时间序列存在如下关系：

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t$$


The diagram shows the equation  $y(t) = g(t) + s(t) + h(t) + \epsilon_t$  with four components enclosed in dashed boxes of different colors. Arrows point from each box to its corresponding label: a purple box for  $g(t)$  points to '趋势成分' (Trend component), a yellow box for  $s(t)$  points to '周期成分' (Seasonal component), a green box for  $h(t)$  points to '节假日成分' (Holiday component), and a blue box for  $\epsilon_t$  points to '噪音成分' (Noise component).

机器学习模型在各方面领域已经展示其强大的拟合能力。可以通过学习历史时间序列数据，让模型去预测接下来的数据值。

模型的输入： $[\mathbf{v}_1, \dots, \mathbf{v}_n]$

模型输出： $[\mathbf{v}_{n+1}, \dots, \mathbf{v}_{n+m}]$

广泛应用于时间序列预测的机器学习模型是多层感知机(Multi-Layer Perceptron, MLP)、长短期记忆(Long Short-Term Memory, LSTM)网络、循环门控单元(Gated Recurrent Unit, GRU)网络和递归神经网络(Recurrent Neural Network, RNN)。

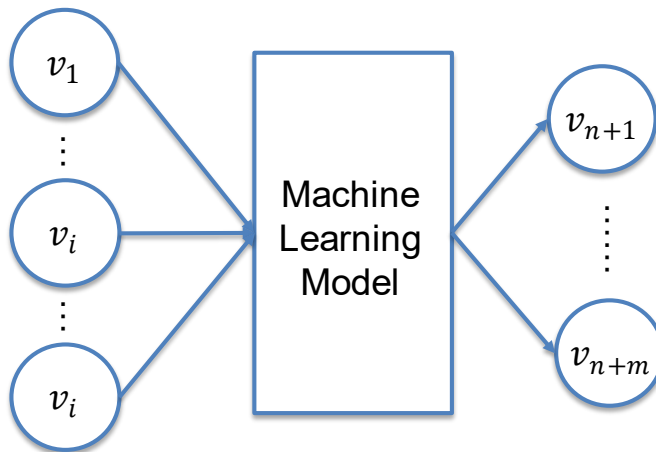


图16: 基于机器学习的时间序列预测框架



---

# 时间序列分类

---

誠樸雄偉  
勵學敦行

- 假设数据产生速率是恒定的，一元时间序列  $X = \{(t_1, x_1), \dots, (t_l, x_l)\}$  可以简写为  $X = [x_1, x_2, \dots, x_{l-1}, x_l]$ ； $M$  元时间序列  $MTS = [X^1, \dots, X^M]$  是由  $M$  个不同的一元时间序列组成， $X^i \in \mathbb{R}^l$ 。
- 数据集  $D = \{(X_1, Y_1), (X_2, Y_2), \dots, (X_N, Y_N)\}$  是  $(X_i, Y_i)$  集合，其中  $X_i$  可以是单变量或多变量时间序列，其中  $Y_i$  作为其对应的 one-hot 标签向量。对于包含  $K$  个类别的数据集，one-hot 标签向量  $Y_i$  是长度为  $K$  的向量，其中每个元素  $j \in [1, K]$  如果  $X_i$  的类别为  $j$ ，则等于 1，否则为 0。

时间序列分类(Time Series Classification, TSC)的任务是在数据集  $D$  上训练分类器，以便尽可能将时间序列从输入空间映射到类标签。

时间序列分类方法有基于  $k$ -NN，基于深度学习和基于图的方法。



# 基于 $k$ -NN的时间序列分类

对于一个测试实例，使用相似性度量方法计算训练数据离该测试实例最相似的 $k$ 个数据，该实例的类标签为最相似的 $k$ 个数据的主导标签。

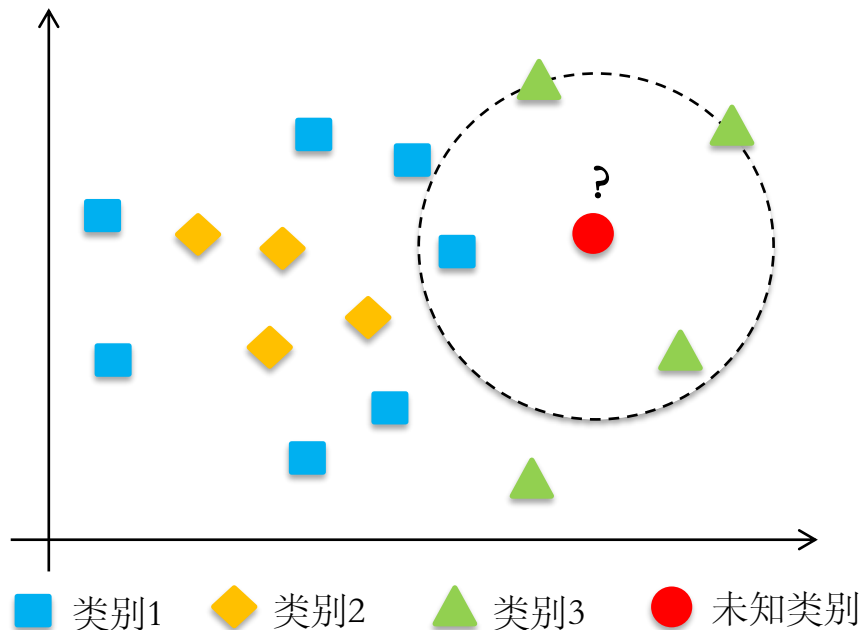


图17: 基于4-NN的时间序列分类示意图

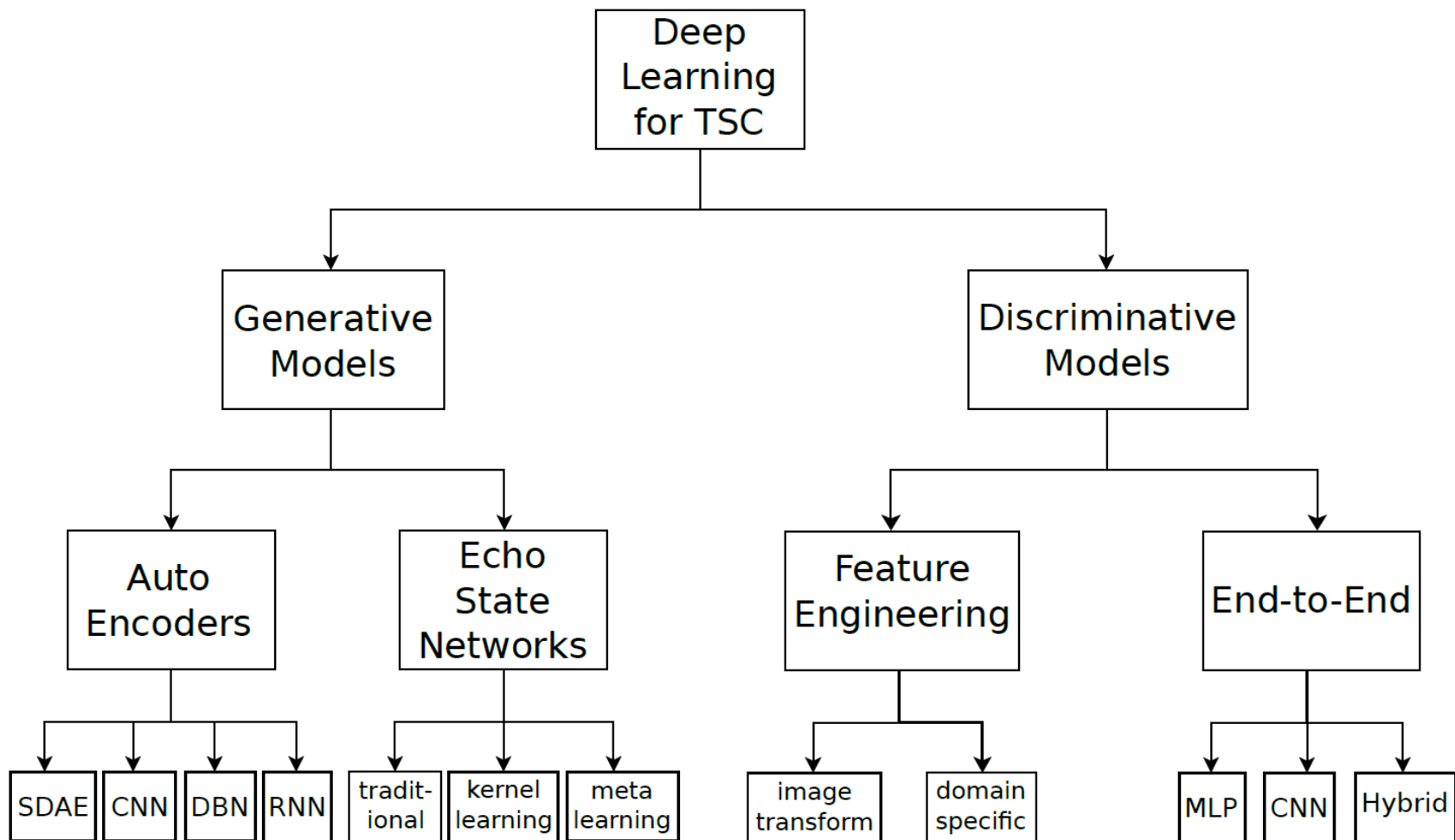


图18: 基于深度学习的时间序列分类方法总览

# 基于深度学习时间序列分类的统一框架

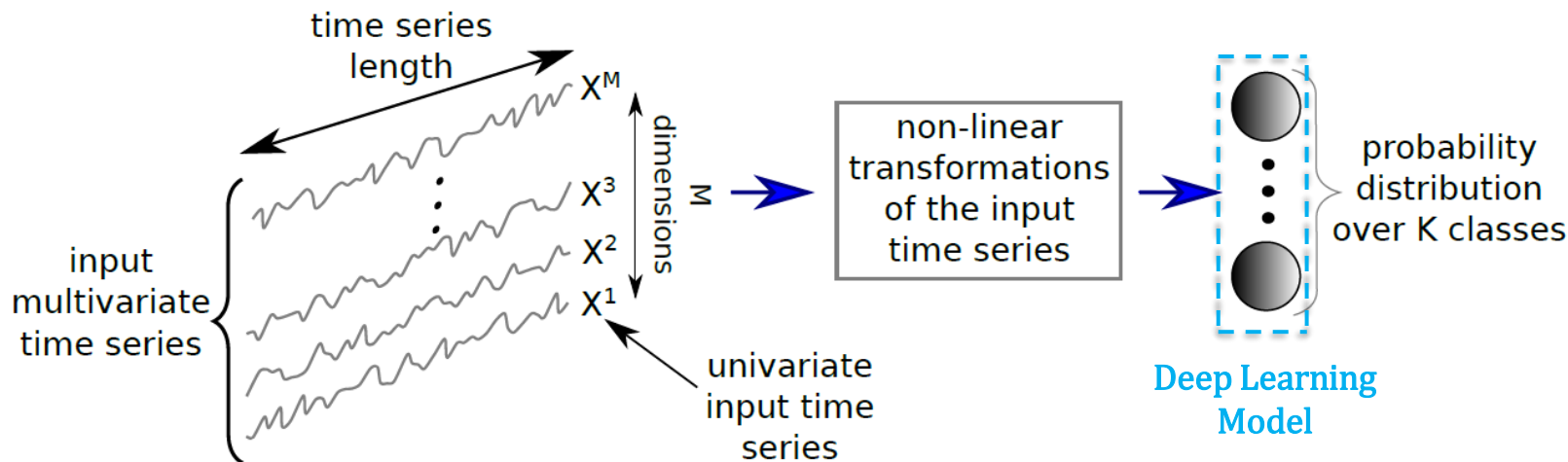


图19: 基于深度学习时间序列分类的统一框架

深度神经网络由 $L$ 层网络构成，每一层都可以将其视为一个函数 $f$ ，给定输入 $\vec{x}$ ，深度神经网络根据以下公式确定预测的类别：

$$f_L(\theta_L, \vec{x}) = f_{L-1}(\theta_{L-1}, f_{L-2}(\theta_{L-2}, \dots, f_1(\theta_1, \vec{x})))$$

其中 $\theta_i$ 表示第 $i$ 层的相关参数， $f_i$ 表示第 $i$ 层表示的函数。

可以选择多层感知机(Multi Layer Perceptron, MLP)，卷积神经网络(Convolutional Neural Networks, CNN)和循环神经网络(Recurrent Neural Network, RNN)作为分类模型。

# 卷积神经网络用于时间序列分类架构

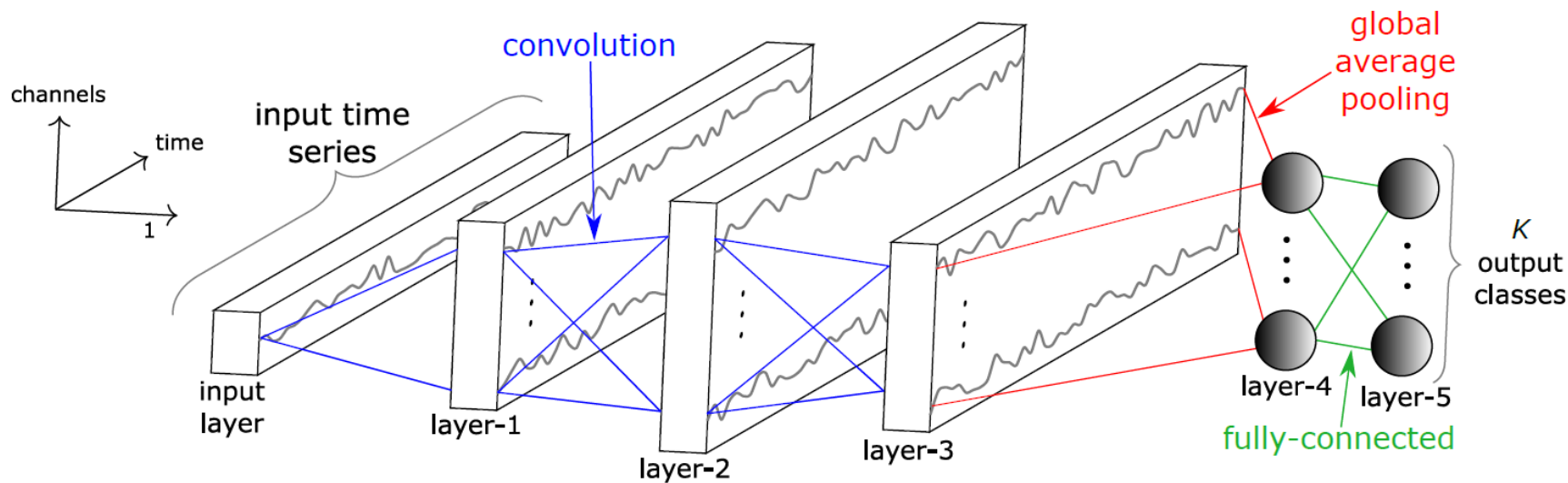


图20: 卷积神经网络用于时间序列分类架构

卷积可以看作是在时间序列上应用和滑动过滤器。过滤器可以看作是时间序列的一般非线性变换。

# 基于图的时间序列分类

从训练和测试实例中构建一个相似图 $G = (V, E)$ ， $V$ 中每一个节点对应于一个训练或测试数据中的实例， $V$ 中每个节点用一条无向边连接，其权重为两个节点的相似度，边集合为 $E$ 。将 $G$ 中有标签的节点想象成染过色，而无标签的的节点尚未染色，于是基于图的时间序列分类对应于“颜色”在图上扩散或传播过程。

基于图的时间序列分类与图半监督学习思想相同。

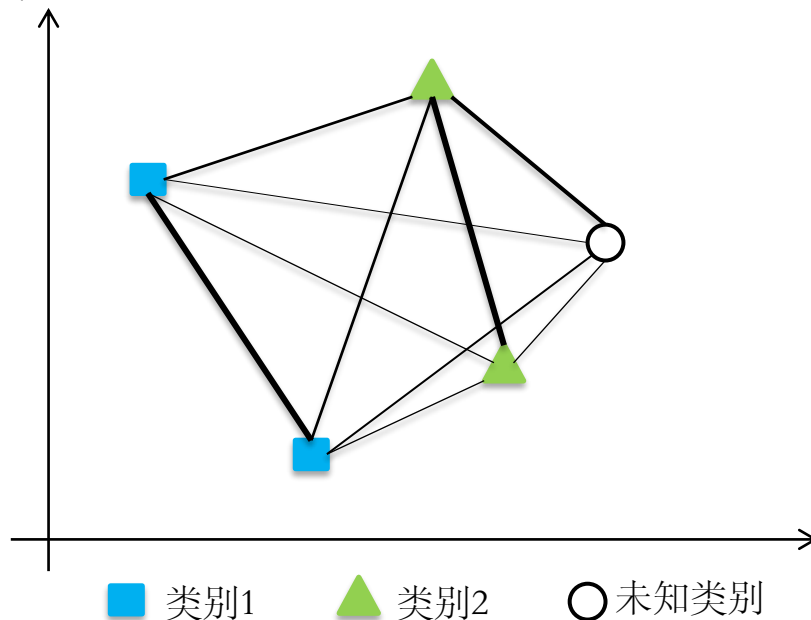


图21: 基于图的时间序列分类，顶点对应于数据实例，边用来表示两个数据实例的相似度，边越“粗”表示相似度越高



---

# 时间序列聚类

---

誠樸雄偉  
勵學敦行

**聚类：**相似数据放入相关或同质的组中，而无需事先了解组的定义。

**时间序列聚类：**给定一个时序数据集  $D = \{F_1, F_2 \dots F_n\}$ , 我们的目标是将其聚类为  $C = \{C_1, C_2 \dots C_k\}$  个簇；每个  $F_i$  只属于1个类，且不同类之间没有交集。相较于常规的静态数据，时序数据具有时间上的依赖性。

时间序列聚类的意义：

- (1) 可以简化时间序列问题的处理难度
- (2) 时间序列聚类是更复杂的数据挖掘算法(如规则发现、索引、分类和异常检测)的必备过程之一
- (3) 将时间序列的簇结构表示为可视化图像可以帮助用户快速理解数据集中的数据结构、簇、异常和其他规则

时间序列聚类和分类的区别在于时间序列有无标签。

- 时间序列数据量大，而传统聚类的计算复杂度较高，导致大型时间序列数据的聚类速度非常慢，空间的占用需求高
- 数据维度高，很多传统聚类算法收敛很慢从而导致算法失效
- 数据噪声较高，非常容易出现异常值，并且存在滞后性，致使序列出现偏离，影响聚类算法设计
- 多序列聚类问题中，序列长度存在差异
- 时间序列相似性度量方法的选择对聚类结果影响很大



- 基于形状：两个时间序列的形状尽可能地匹配，要考虑到时间轴的拉伸和收缩。这种方法也被称为基于原始数据的方法，因为它通常直接处理原始时间序列数据
- 基于特征：原始时间序列被转换成低维特征向量（例如基于滑动窗口做特征衍生，时间序列分解等），然后采用传统的聚类算法对提取的特征向量进行聚类
- 基于模型：每个时间序列被转换为模型参数，然后选择合适的模型距离和聚类算法，并将其应用于提取的模型参数

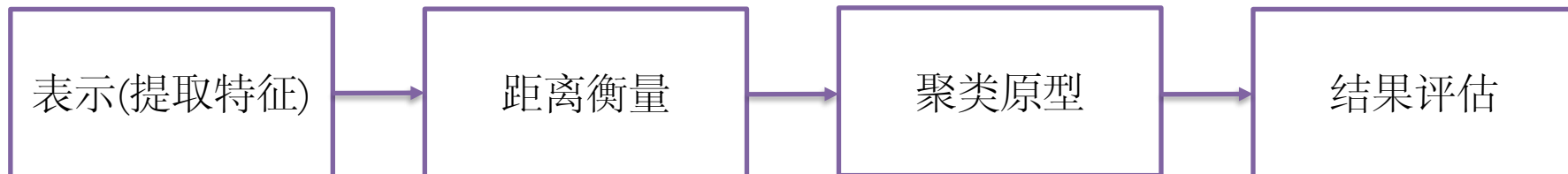
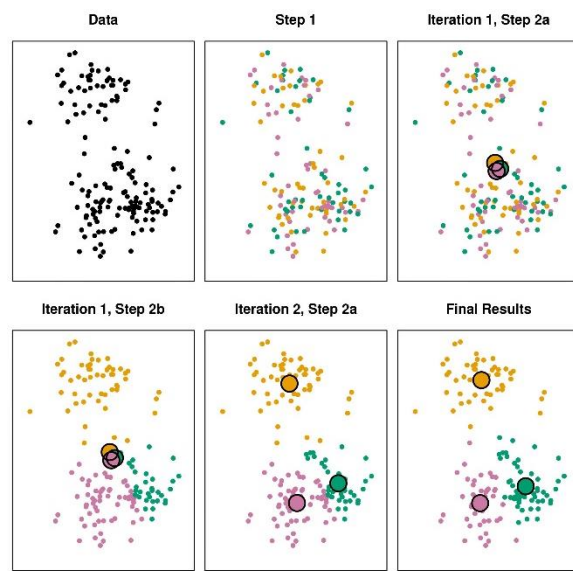


图22: 时间序列聚类步骤

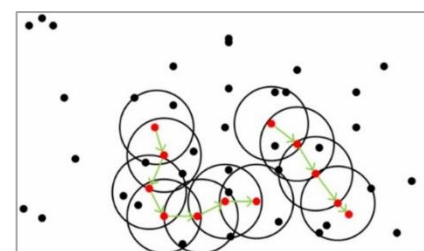
说明:

- (1) 时序数据进行特征提取的意义在于降低了时间序列数据维度, 从而降低了计算开销, 加速了聚类的收敛速度、有效减小数据中的噪声。
- (2) 降维会导致丢失部分信息, 可能会影响降维的质量, 因此聚类需要综合考量速度和质量。
- (3) 聚类原型就是一些聚类的簇心, 原型的质量会对聚类的质量有较大的影响。

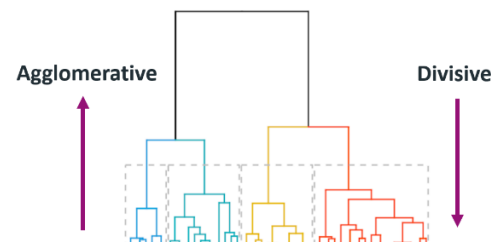
- 分区聚类：初始设定K个簇，然后将数据按照距离放入不同的簇，常见算法是K-means算法
- 密度聚类：假设聚类结构能通过样本分布的紧密程度确定，从样本密度的角度来考察样本之间的可连接性，并给予可连接样本不断扩展聚类簇获得最后的聚类结果，常见的如DBSCAN
- 层次聚类：在不同层次对数据集进行划分，形成树形的聚类结构，可以“自底向上”聚合，也可以“自顶向下”拆分，常见的如Agglomerative



(a)K-means算法



(b)DBSCAN算法



(c)Agglomerative算法

图23：传统时间序列聚类算法示意图

主要思想是用模型提取时间序列的特征，对降维后的特征再使用一些传统的聚类算法进行聚类。采用这种框架有两种处理方式：

- (1) 多步聚类：先训练提取特征的 *encoder – decoder* 部分，再用提取好的特征训练聚类部分
- (2) 联合聚类： *encoder – decoder* 和聚类同时训练，整个模型的损失由重构损失和聚类损失两部分组成

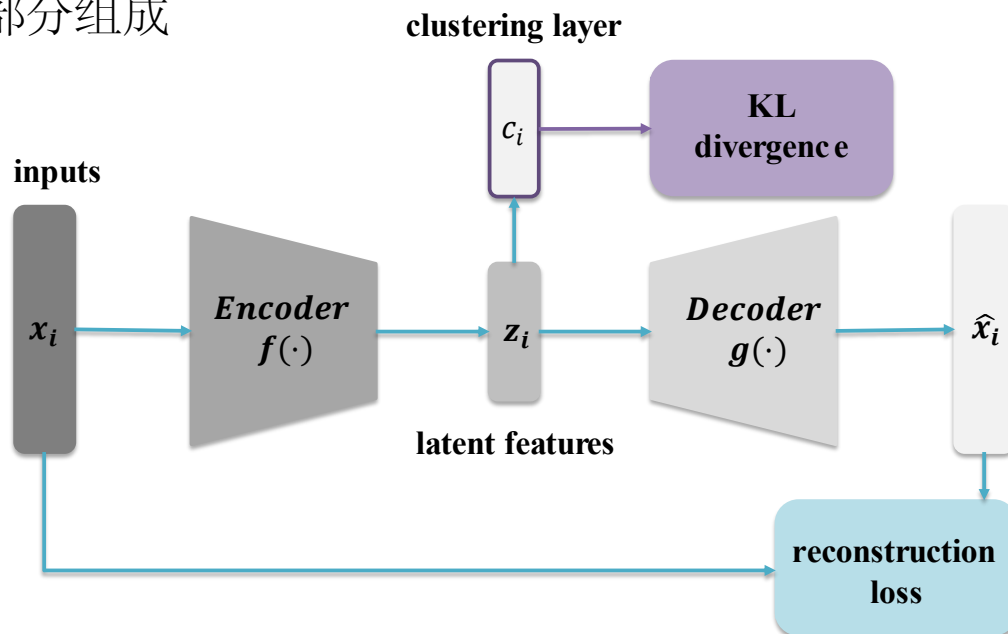


图24：基于深度学习的时间序列聚类框架



---

# 时间序列异常检测

---

誠樸雄偉  
勵學敦行

时间序列的异常检测问题也可以以两种不同的方式定义：

- 点异常：给定的时间戳上的时间序列值突然变化
  - 例子：股票在某一时刻价格暴涨/暴跌，之后恢复正常情况
  - 检测方法：基于预测模型、基于相似度量
  
- 形状异常：一个连续窗口中的数据点的连续模式
  - 例子：ECG序列中，不规则心跳
  - 检测方法： $k$ 近邻法

基于预测模型的点异常检测跟时间序列预测问题密切相关，具体步骤如下：

Step 1: 确定时间序列在每个时间戳上的预测值(任何一个时间序列预测模型都行)，将第 $r$ 个时间戳 $t_r$ 上的预测值表示为 $\overline{W}_r$ 。

Step 2: 计算时间序列的预测偏差 $\overline{\Delta}_1, \dots, \overline{\Delta}_r, \dots$ ;

Step 3: 计算时间序列的预测偏差的均值 $\mu$ 和标准差 $\sigma$ ;

Step 4: 计算标准化的偏差 $\delta_r = \frac{\overline{\Delta}_r - \mu}{\sigma}$ 。

由此产生的偏差基本上等于正态分布的 $Z$ 值。该方法为时间序列提供了一个连续的异常得分警报级别。

基于预测模型的点异常检测本质上是依据n-sigma准则。

## □ 基于距离度量

- 对于每个点计算它的 $k$ 邻近距离，然后按照距离降序排序，前 $m$ 个点认为是异常点

## □ 基于LOF度量

- 对数据集每个点计算一个离群因子(Local Outlier Factor, LOF),通过判断LOF是否接近于1来判定是否是离群因子。若LOF远大于1，则认为是离群因子，接近于1，则是正常点

## □ 基于孤立森林(Isolation Forest)

- 基于集成学习，适用于连续数据的异常检测，通过多颗iTree形成森林来判定是否有异常点

## □ 基于神经网络

- 通过寻找神经网络的重构误差(使用相似性度量方法评估)来区分正常点和异常点



表5: 基于相似度量的点异常检测方法对比

方法	优点	缺点
基于距离度量	简单, 无分布假设	只能找到点不能找簇, 仅适合全局异常
基于LOF度量	可找局部异常	高维数据计算量很大
基于孤立森林	可分布式部署, 适用于大数据集, 计算速度快	检测全局且不适用于高维数据
基于神经网络	适合高维数据, 效果好	数据需要具有全局相关性

假设在长度为 $W$ 的窗口上进行异常检测分析， $k$ 近邻法使用欧氏距离来评估异常得分，然后根据报告时间序列数据点中具有与众不同的形状窗口。 $k$ 近邻异常检测算法步骤如下：

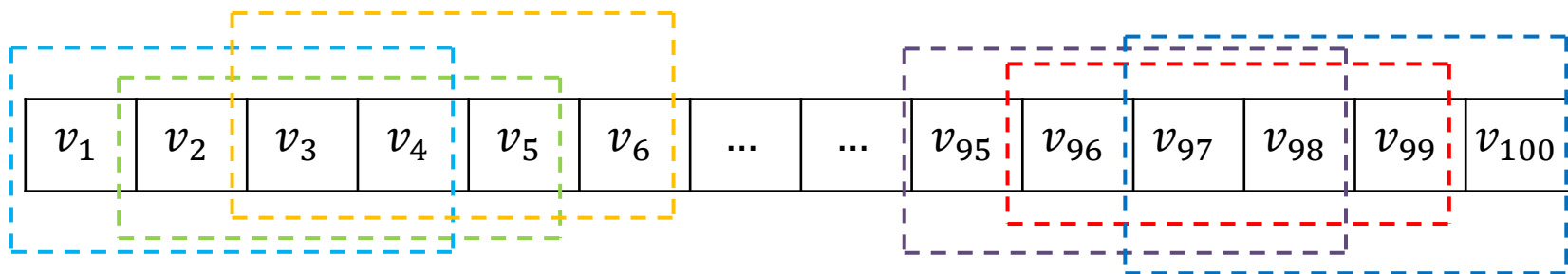
- Step 1: 使用滑动窗口法从时间序列中获取所有长度为 $W$ 的窗口；
- Step 2: 对每一个获取的窗口，计算它与其他非重叠窗口的欧氏距离；
- Step 3: 将与其 $k$ 个最近距离最大的窗口报告为异常。

$k$ 近邻算法不足：算法复杂度与数据点数量的平方成正比

解决方案：剪枝

# $k$ 近邻法进行形状异常举例

假设  $W = 4, k = 2, n = 100$ .



	No.1-NN	Distance	No.2-NN	Distance
$w_1$	$w_{24}$	3.6	$w_{25}$	7.4
$w_2$	$w_{25}$	9.5	$w_{96}$	11.2
$w_3$	$w_{97}$	28.3	$w_{56}$	34.7
...				
...				
$w_{97}$	$w_{68}$	5.9	$w_{49}$	8.9

异常片段

图25:  $k$ 近邻法进行形状异常检测例子



---

# 时间序列数据库

---

誠樸雄偉  
勵學敦行

时间序列数据库 (Time Series Database, TSDB) 是为处理时间序列数据而优化的数据库。时间序列数据库可以分成两类，基于现有的数据库改写的，或者专为时间序列数据读写开发的新数据库。

**Q:** 为什么需要时间序列数据库?

**A:** 数据库是在特定的问题场景需求下对性能折中，没有在各种场景下都适用的数据库。业界需要一种低成本、支持快速写入，快速查询和分析等功能的数据库。

## □ 写入

- 写入持续、平稳、高吞吐
- 写多读少

## □ 查询

- 按照时间范围读取
- 最近的数据被读取的概率高、历史数据粗粒度查询的概率高
- 多精度、多维度查询

## □ 存储

- 量大
- 冷热分离
- 时效性
- 多精度数据存储

# 时间序列数据库产品诞生时间表

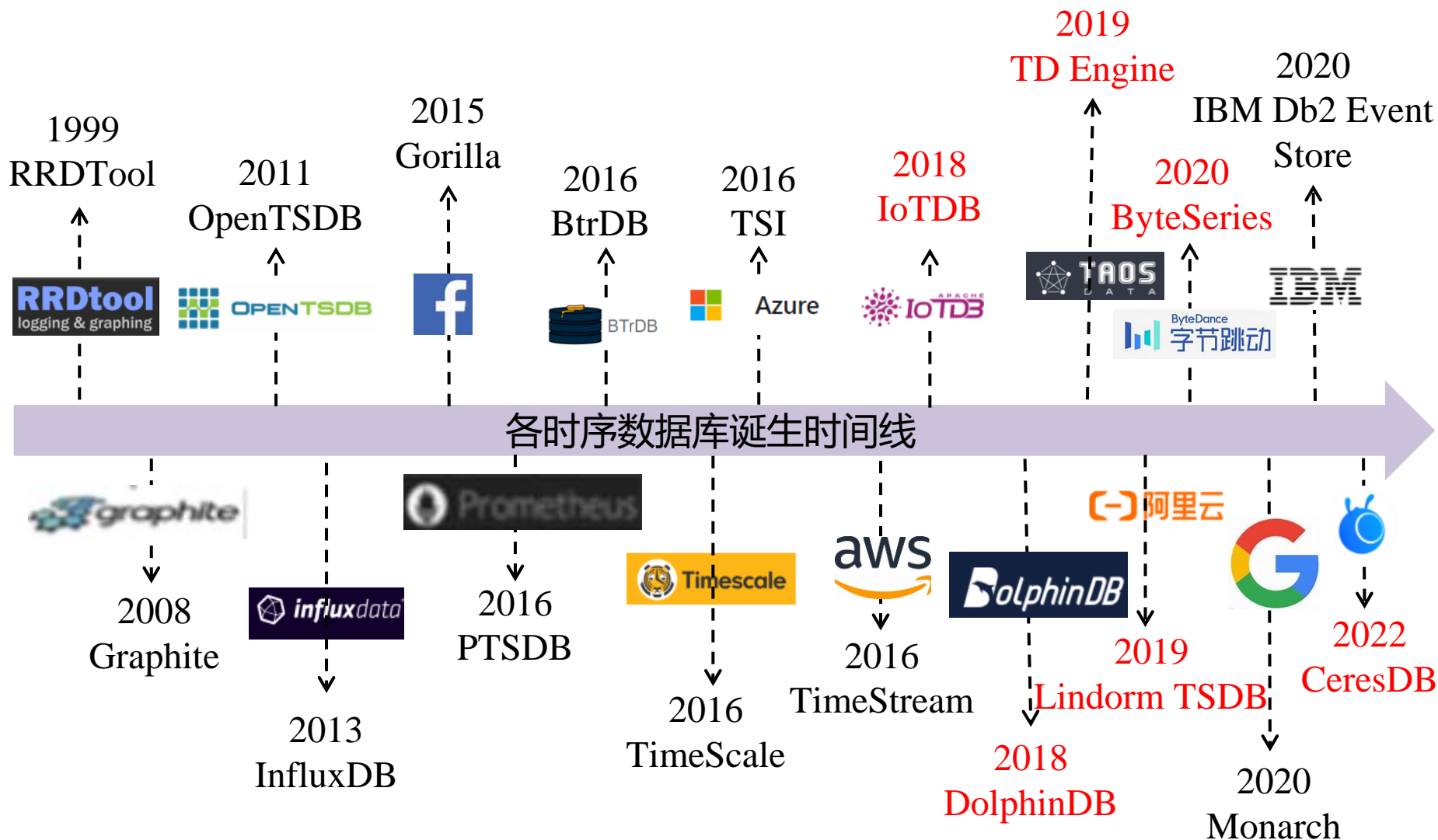


图26：部分时间序列数据库产生诞生时间表

# 时间序列数据库核心技术：LSM-Tree

传统关系型数据库存储采用的都是 B Tree/B+ Tree，这是由于其在查询和顺序插入时有利于减少寻道次数的组织形式。由于时间序列的特性，时间序列数据库对写速率要求较高，采用的是类LSM-Tree技术存储。

LSM-Tree核心思想就是利用顺序写来提高写性能，但因为LSM-Tree分层设计(此处分层是指的分为内存和文件两部分)，降低了LSM-Tree读性能（LSM-Tree是通过牺牲小部分读性能换来高性能写的）。

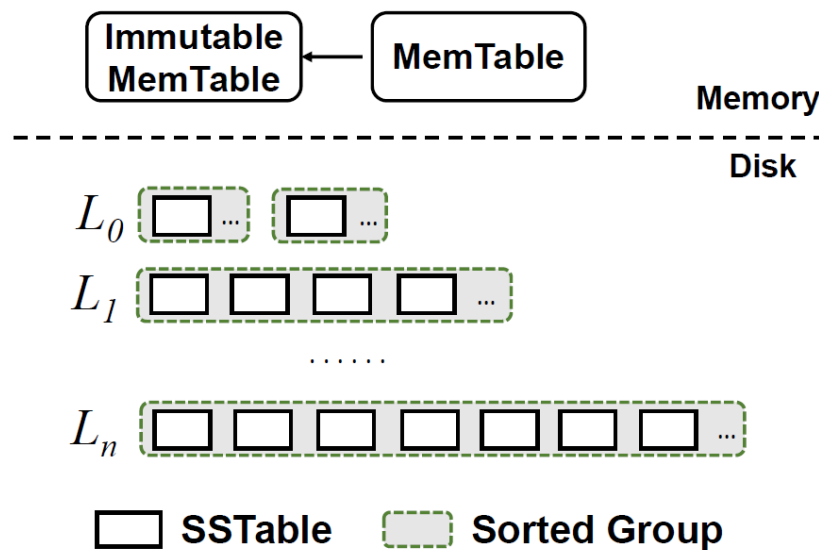


图27：传统LSM-Tree架构



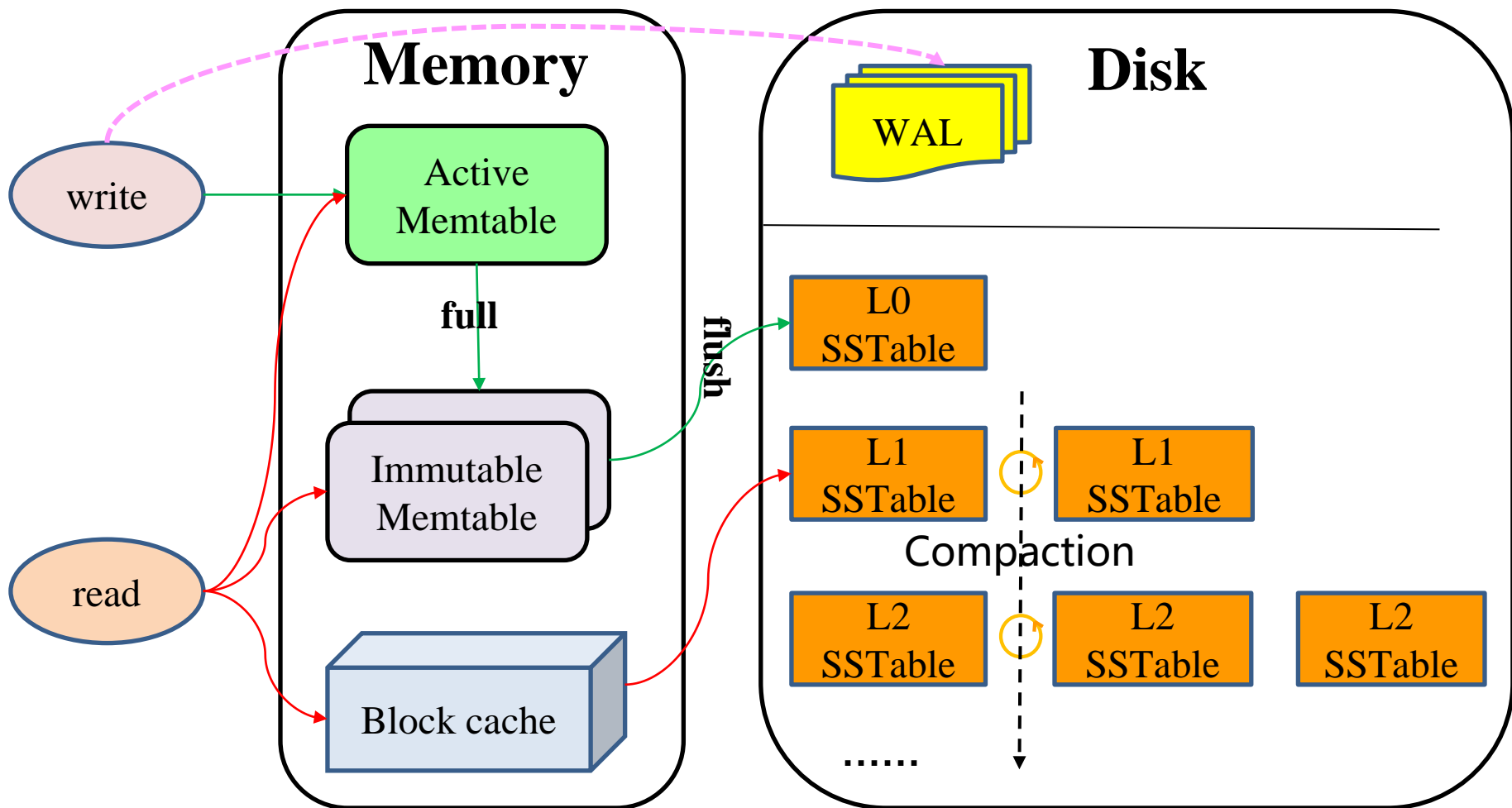


图28: LSM-Tree读写流程示意图

- ❑ **读放大**: 读取数据时实际读取的数据量大于真正的数据量。例如在LSM-Tree中需要先在MemTable查看当前key是否存在, 如果不存在则需要继续从各层SSTable中寻找。
- ❑ **写放大**: 写入数据时实际写入的数据量大于真正的数据量。例如在LSM-Tree中写入时可能触发Compact操作, 导致实际写入的数据量远大于该key的数据量。
- ❑ **空间放大**: 数据实际占用的磁盘空间比数据的真正大小更多。因为可能key即存在内存中, 又存在磁盘各层SSTable中, 这个key是冗余存储的。

LSM-Tree牺牲了读速率来换取写速率，那么如何尽可能保证读速率够高呢？

Key <sub>1</sub>	Value <sub>1</sub>	Key <sub>2</sub>	Value <sub>2</sub>	Key <sub>3</sub>	Value <sub>3</sub>	....	....
------------------	--------------------	------------------	--------------------	------------------	--------------------	------	------

LSM-Tree的SSTable

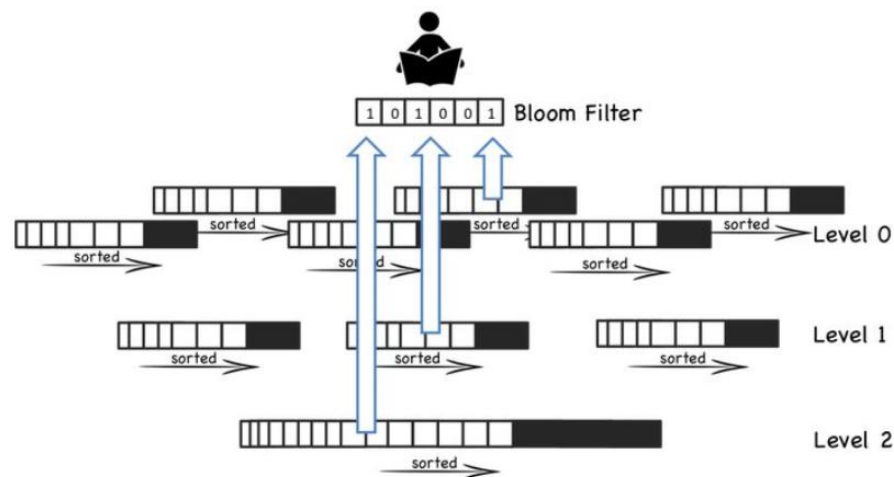
由于内存读取数据比磁盘快很多，那么加速查询主要要优化磁盘的数据查询。

方案：使用key的索引表以及Bloom Filter来加快key的查找。

## Index

Key <sub>1</sub>	Offest <sub>1</sub>
Key <sub>2</sub>	Offest <sub>2</sub>
Key <sub>3</sub>	Offest <sub>3</sub>
....	....

(a)使用key的索引表加速查询

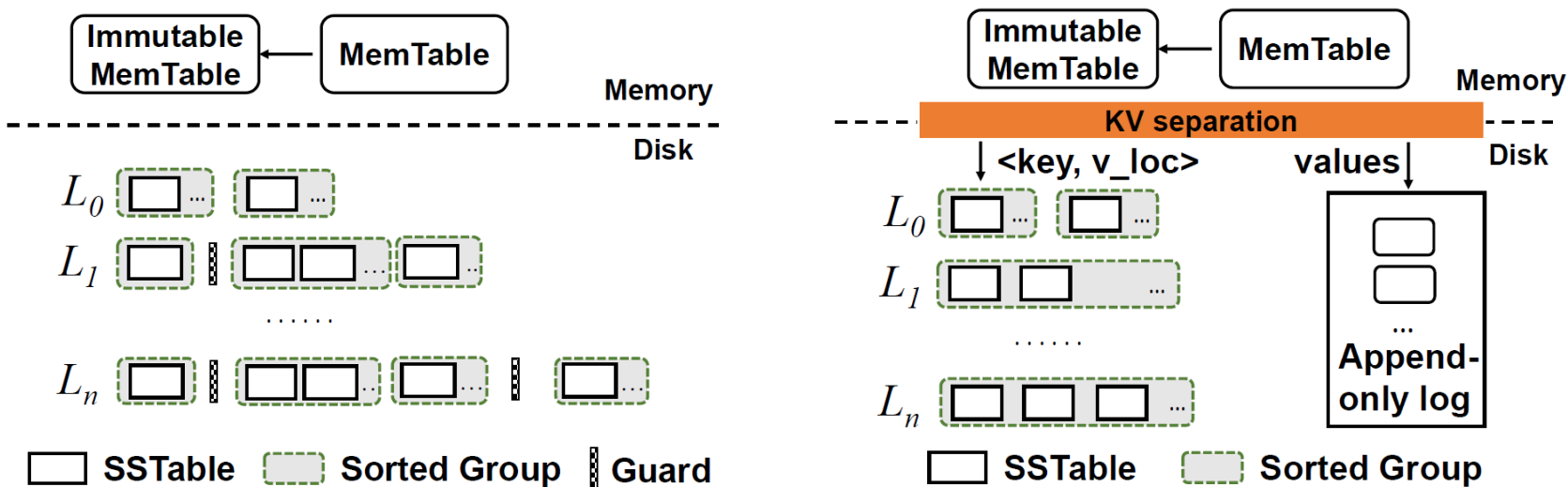


(b)使用Bloom Filter避免不必要的读

图29: LSM-Tree查询加速策略

# 缓解LSM-Tree写放大策略

- 各层分段，部分有序
- 键值分离



(a) 每层部分有序

(b) 键值分离

图30: LSM-Tree写放大缓解策略

- 动态调整每一层数据大小，让数据尽可能compact到最高层

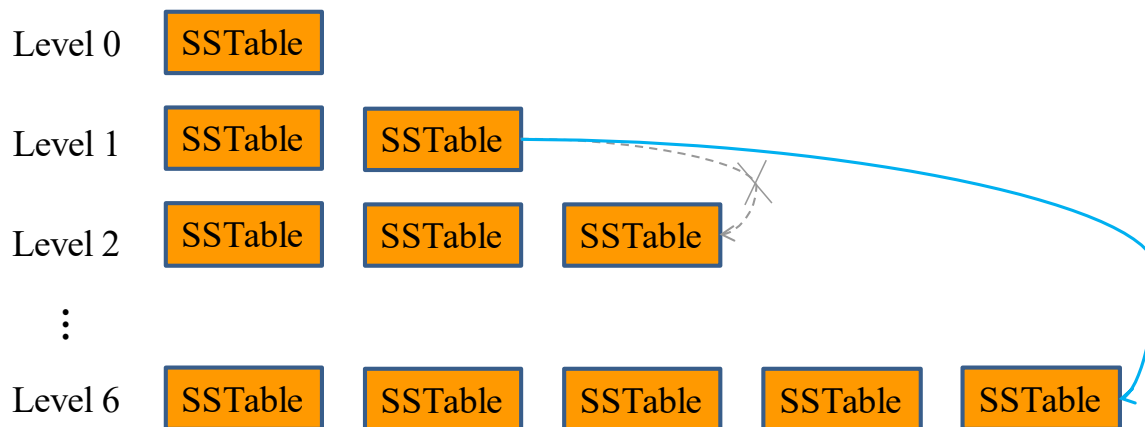


图31: LSM-Tree空间放大缓解策略

# 时序数据库产品IoTDB简介

Apache IoTDB（物联网数据库）是一体化收集、存储、管理与分析物联网时序数据的软件系统。Apache IoTDB 采用轻量式架构，具有高性能和丰富的功能，并与Apache Hadoop、Spark和Flink等进行了深度集成，可以满足工业物联网领域的海量数据存储、高速数据读取和复杂数据分析需求。

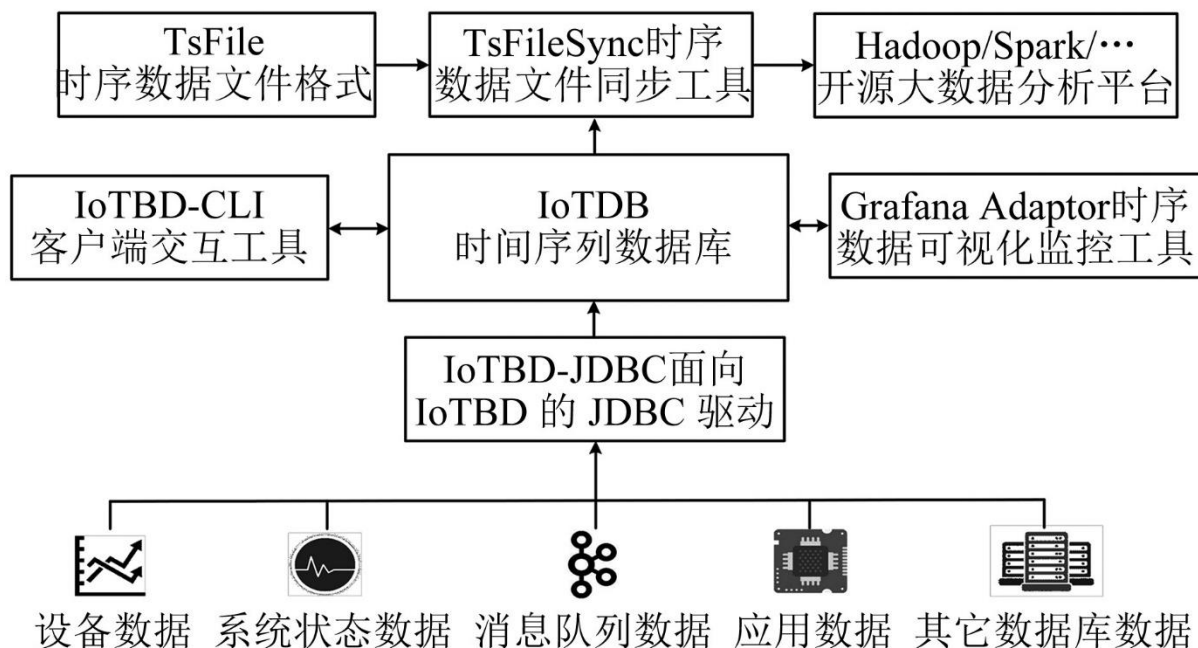


图32: IoTDB应用架构



---

# 时间序列智能运维

---

誠樸雄偉  
勵學敦行

智能运维(Artificial Intelligence for IT Operations, AIOps), 指的是将人工智能应用于运维领域, 通过机器学习从而发现和解决传统的自动化运维无法解决的问题。

智能运维 = 运维场景 + 智能技术

时间序列运维应用: 预测、异常检测、告警、故障诊断与根因分析……

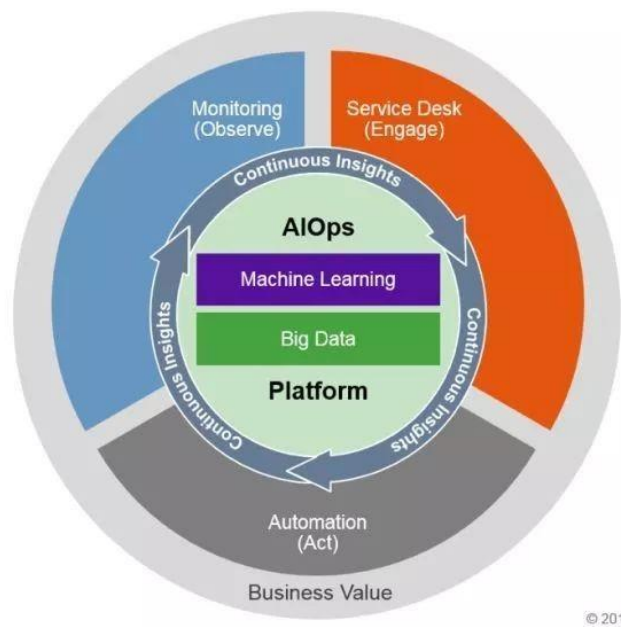


图33: Gartner定义AIOps概念图, 来源: Gartner(2017年8月)



从消费互联网到产业互联网，数字化转型已成为企业发展的必修课。

中国在2020年底发布的“十四五规划建议”中**6次**提及“数字化”，对政府、数字经济、数字中国、金融、服务业、公共文化等不同方面均提出了要求，其中着重提到要发展数字经济，推进数字产业化和产业数字化，加强数字社会、数字政府建设，提升公共服务、社会治理等数字化智能化水平。

2020年中国数字经济规模达到39.2万亿元，占GDP比重为**38.6%**。

中国信息通信研究院

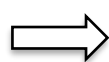
2020年我国数字经济核心产业增加值占GDP的比重达**7.8%**。

国家网信办

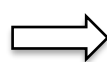
2024年中国IT预算中用于数字化转型这一占比将超过**70%**。

IDC

数字化进程越深厚



IT系统越来越重要



IT系统的运维也就越来越重要

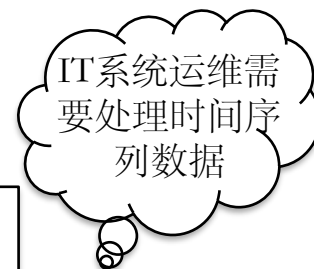


图34：数字经济与智能运维关系

运维数据:

- ❑ 指标(Metrics): 可聚合的逻辑计量单元
- ❑ 日志(Logging): 对离散的不连续的事件一种记录
- ❑ 追踪(Tracing): 调用链信息

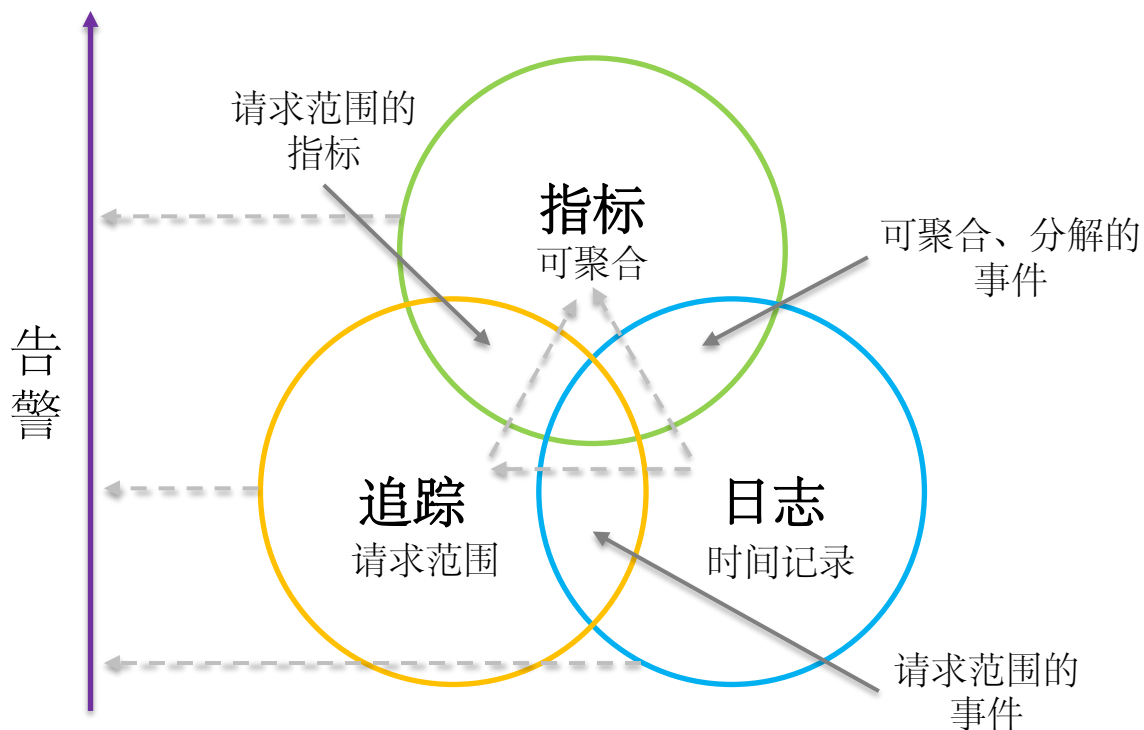


图35: 指标、日志、追踪及其关系

- ❑ 体量大：国内企业5000万指标数据的实时检测
- ❑ 缺失补全：对数据确实进行线性/均值/特定值填充等插值方式
- ❑ 峰谷潮：系统采集数据量有时突然激增，采集数据量有时突然减少
- ❑ 乱序容忍：受采集设备、网络传输等影响，数据出现10%左右的乱序
- ❑ 粒度齐整：受限于采集器，数据的采集时间不够精确
- ❑ 单点爆炸：受采集设备影响，某时间戳爆炸式的出现多个数据点

面对运维数据，行业的一般解决方案：

- (1) 自研数据库，如servicenow使用自研数据库MetricBase
- (2) 与开源数据库融合，如Cloudwise使用的是DODB+IoTDB

## 性能优势

- 高频数据写入和查询
- 多种历史数据压缩方式节省成本

## 功能优势

- 支持异常数据处理场景
- 数据降采样提升查询响应速度
- 多种操作提升预处理效果
- 可自定义计算方式及保存计算结果
- 兼容大数据分析工具
- 提供可视化工具展示数据

单机1万条指标，100亿个点，5线程并发  
性能对比图

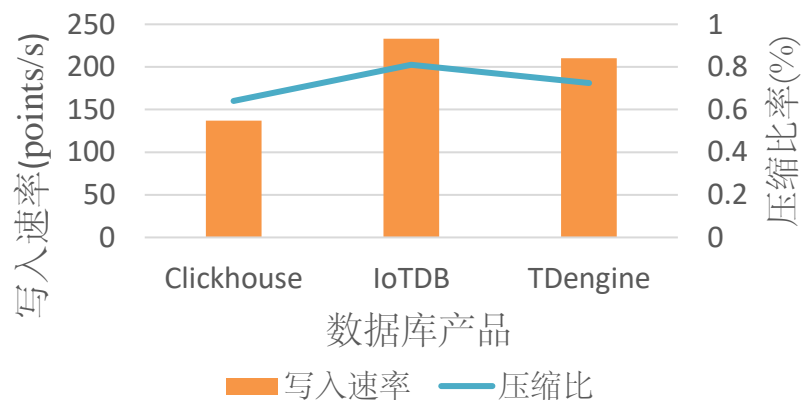


图36：不同时间序列数据库写入性能和压缩性能对比，数据来源文献[25]

表6：相关挑战的IoTDB解决方案

挑战	IoTDB解决方案
缺失值补全	提供零值填充/线性填充
乱序插入	数据排序后再存储
单点爆炸	单个时间点只允许写一次
粒度齐整	根据数据区间进行数据粒度规整

## 例子1：城市轨道交通车辆运维系统

**业务痛点：**城市轨道交通车辆智能运维系统需要实现从数据采集、数据存储到数据分析、数据展示的全流程、全功能的覆盖；且系统内的数据具有变量多、周期短、变化小、时效性强等特点，这对时间序列数据库系统中各功能模块的性能提出了较高要求，具体要求如下：

- 毫秒级实时数据接收
- TB级数据存储
- 实时监控
- 便捷计算统计

基于IoTDB时序数据库构建城轨车辆智能运维系统总体架构共分为3层，包括数据源层、数据存储层和数据应用层。

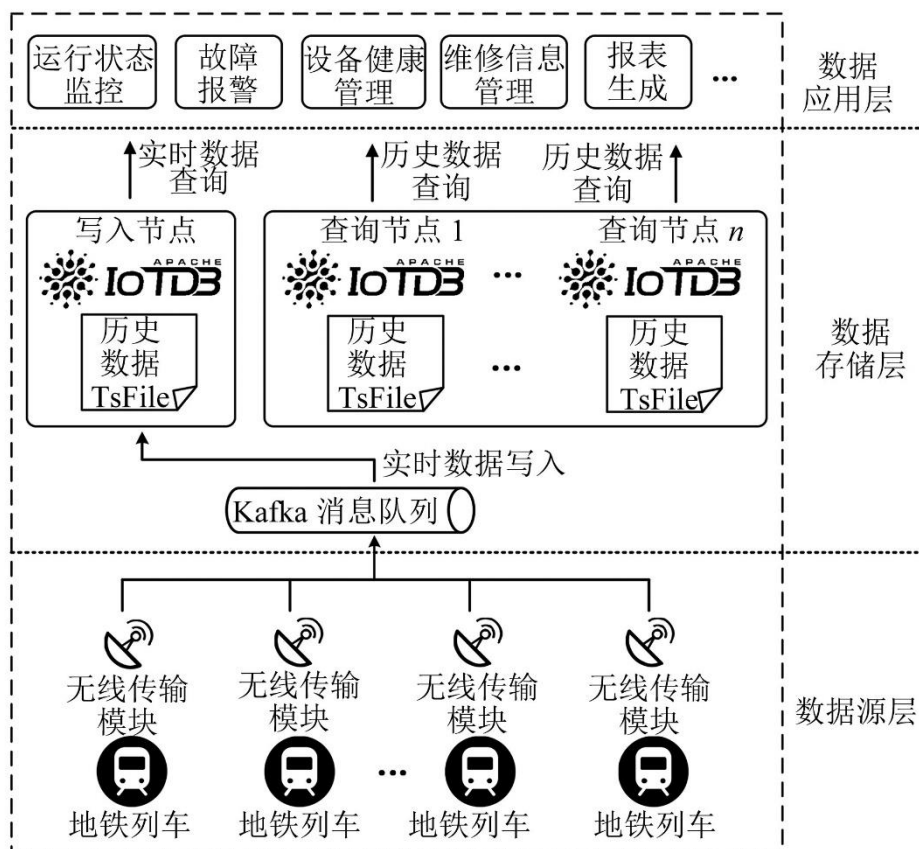


图37：基于IoTDB时序数据库构建城轨车辆智能运维系统架构

# 基于IoTDB的智能运维举例(二)

## 例子2: 智慧养老系统

相关介绍: 智慧养老系统可实现对用户各种生命体征以及智能家居(包括血压, 心率, 呼吸, 体动、灯控、报警等)相关数据的实时采集

业务痛点: 数据量巨大, 设备存储压力大; 数据产生速率快, 而数据分析慢。



图38: 智慧养老系统采集的部分数据

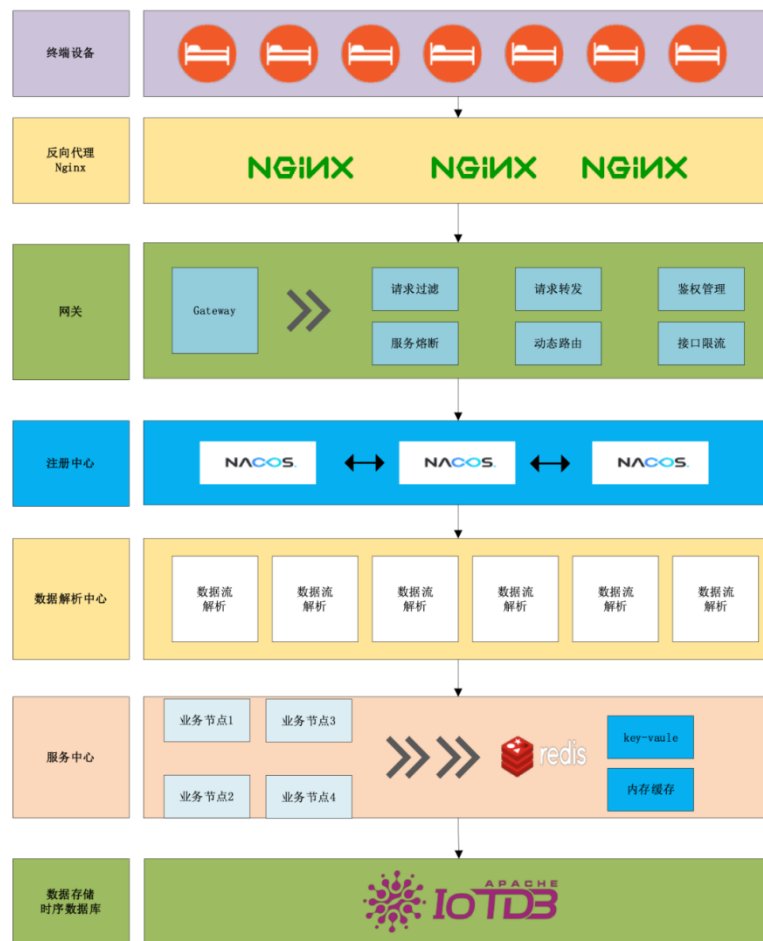


图39: 基于IoTDB时序数据库智慧养老系统架构

- [1] Charu C. Aggarwal. Data Mining[M]. Beijing: China Machine Press. 2020.
- [2] Berndt D J, Clifford J. Using dynamic time warping to find patterns in time series[C]//KDD workshop. 1994, 10(16): 359-370.
- [3] Lin J, Keogh E, Wei L, et al. Experiencing SAX: a novel symbolic representation of time series[J]. Data Mining and knowledge discovery, 2007, 15(2): 107-144.
- [4] Kondylakis H, Dayan N, Zoumpatianos K, et al. Coconut: sortable summarizations for scalable indexes over static and streaming data series[J]. The VLDB Journal, 2019, 28(6): 847-869.
- [5] Shieh J, Keogh E. iSAX: indexing and mining terabyte sized time series[C]//Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. 2008: 623-631.
- [6] Camerra A, Palpanas T, Shieh J, et al. isax 2.0: Indexing and mining one billion time series[C]//2010 IEEE International Conference on Data Mining. IEEE, 2010: 58-67.
- [7] Jafari O, Maurya P, Nagarkar P, et al. A survey on locality sensitive hashing algorithms and their applications[J]. arXiv preprint arXiv:2102.08942, 2021.
- [8] Datar M, Immorlica N, Indyk P, et al. Locality-sensitive hashing scheme based on p-stable distributions[C]//Proceedings of the twentieth annual symposium on Computational geometry. 2004: 253-262.
- [9] Zheng B, Zhao X, Weng L, et al. PM-LSH: a fast and accurate in-memory framework for high-dimensional approximate NN and closest pair search[J]. The VLDB Journal, 2021: 1-25.
- [10] Faloutsos C, Gasthaus J, Januschowski T, et al. Forecasting big time series: old and new[C]. Proceedings of the VLDB Endowment, 2018, 11(12): 2102-2105.
- [11] Faloutsos C, Gasthaus J, Januschowski T, et al. Classical and contemporary approaches to big time series forecasting[C]//Proceedings of the 2019 international conference on management of data. 2019: 2042-2047.
- [12] Jin X, Park Y, Maddix D, et al. Domain adaptation for time series forecasting via attention sharing[C]//International Conference on Machine Learning. PMLR, 2022: 10280-10297.
- [13] Chen Y, Segovia I, Gel Y R. Z-GCNets: time zigzags at graph convolutional networks for time series forecasting[C]//International Conference on Machine Learning. PMLR, 2021: 1684-1694.
- [14] Taylor S J, Letham B. Forecasting at scale[J]. The American Statistician, 2018, 72(1): 37-45.



- [15] Ismail Fawaz H, Forestier G, Weber J, et al. Deep learning for time series classification: a review[J]. Data mining and knowledge discovery, 2019, 33(4): 917-963.
- [16] Ma Q, Zheng J, Li S, et al. Learning representations for time series clustering[C]. 33rd Conference on Neural Information Processing Systems(NeurIPS'19) , 2019.
- [17] Aghabozorgi S, Shirkhorshidi A S, Wah T Y. Time-series clustering—a decade review[J]. Information systems, 2015, 53: 16-38.
- [18] Alghushairy O, Alsini R, Soule T, et al. A review of local outlier factor algorithms for outlier detection in big data streams[J]. Big Data and Cognitive Computing, 2020, 5(1): 1.
- [19] Liu F T, Ting K M, Zhou Z H. Isolation forest[C]//2008 eighth IEEE international conference on data mining. IEEE, 2008: 413-422.
- [20] Cheng Z, Zou C, Dong J. Outlier detection using isolation forest and local outlier factor[C]//Proceedings of the conference on research in adaptive and convergent systems. 2019: 161-168.
- [21] Li Y, Liu Z, Lee P P C, et al. Differentiated Key-Value Storage Management for Balanced I/O Performance[C]//2021 USENIX Annual Technical Conference (USENIX ATC'21). 2021: 673-687.
- [22] Dong S, Kryczka A, Jin Y, et al. Evolution of Development Priorities in Key-value Stores Serving Large-scale Applications: The RocksDB Experience[C]//19th USENIX Conference on File and Storage Technologies (FAST'21). 2021: 33-49.
- [23] Wang C, Huang X, Qiao J, et al. Apache IoTDB: time-series database for internet of things[C]. Proceedings of the VLDB Endowment, 2020, 13(12): 2901-2904.
- [24] Apache IoTDB. “<https://iotdb.apache.org/zh/>.”
- [25] 张博.智能运维场景中的时序数据库选型与挑战. “<https://xie.infoq.cn/article/bc49ca014cb2fc3c7c7644aef>”.
- [26] 姜仕军,徐晓晨,徐燕芬,杜广林.IoTDB物联网数据库在城市轨道交通车辆智能运维系统中的应用[J].城市轨道交通研究,2021,024(009):207-211.
- [27] 怡养科技.Apache IoTDB 在智慧养老家庭设备上的落地应用, 节约99%存储成本. “<https://cloud.tencent.com/developer/article/2004234>”.

