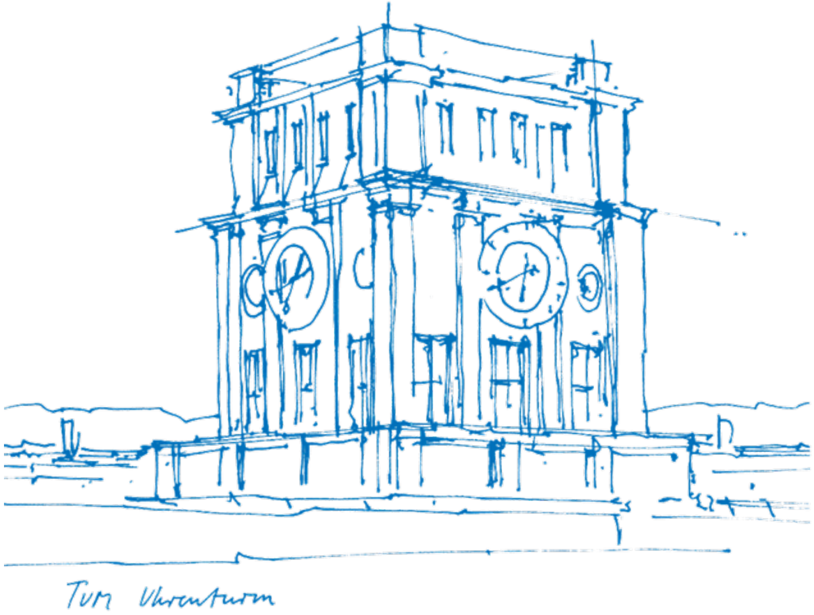
Lehrstuhl III: Datenbanksysteme
Lehrstuhl XXV: Data Science and Engineering
Fakultät für Informatik
Technische Universität München

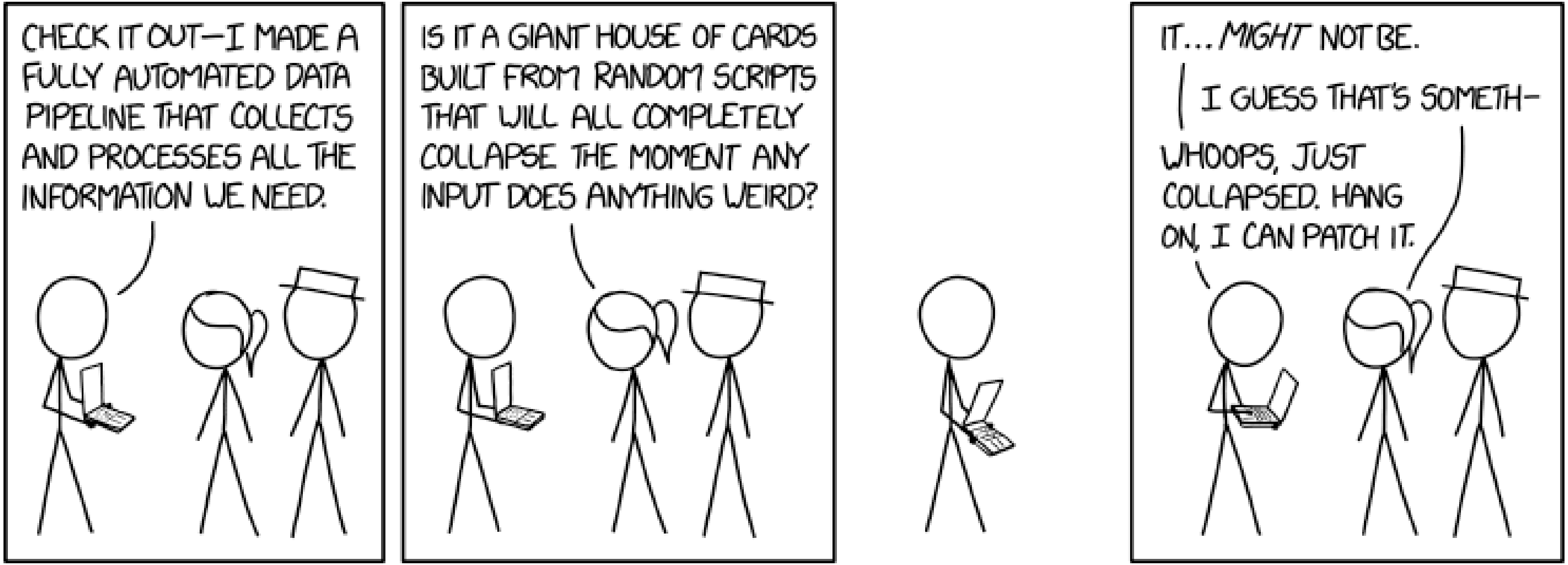
Recursive SQL for Data Mining

Maximilian E. Schüle, Alfons Kemper, Thomas Neumann
Copenhagen, Denmark, July 7, 2022



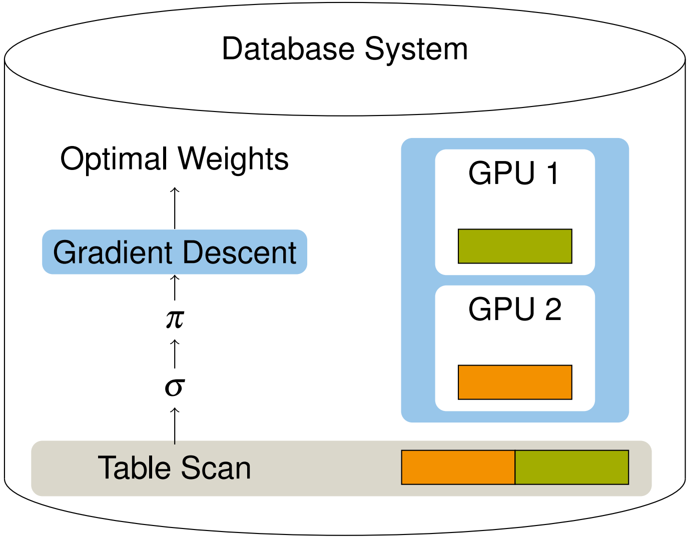
In-Database Machine Learning: Problem

xkcd.org #2054 CC BY-NC 2.5



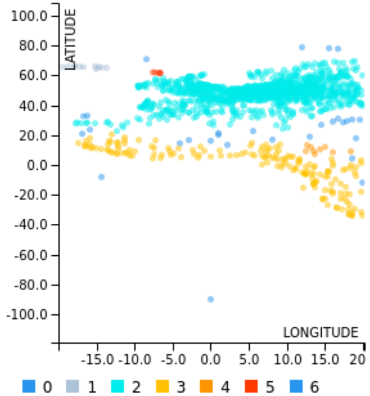
In-Database Machine Learning: Solution

In-Database Machine Learning



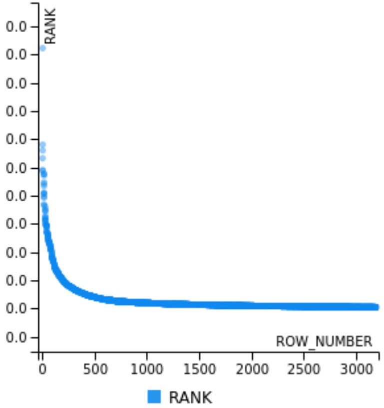
- SQL sufficient for machine learning (ML)
  - Turing-complete with recursive tables
- Idea
  - Data preprocessing using SQL
  - No need for data extraction out of a database system

Structure



Clustering

k-Means, DBSCAN



Graph Analysis

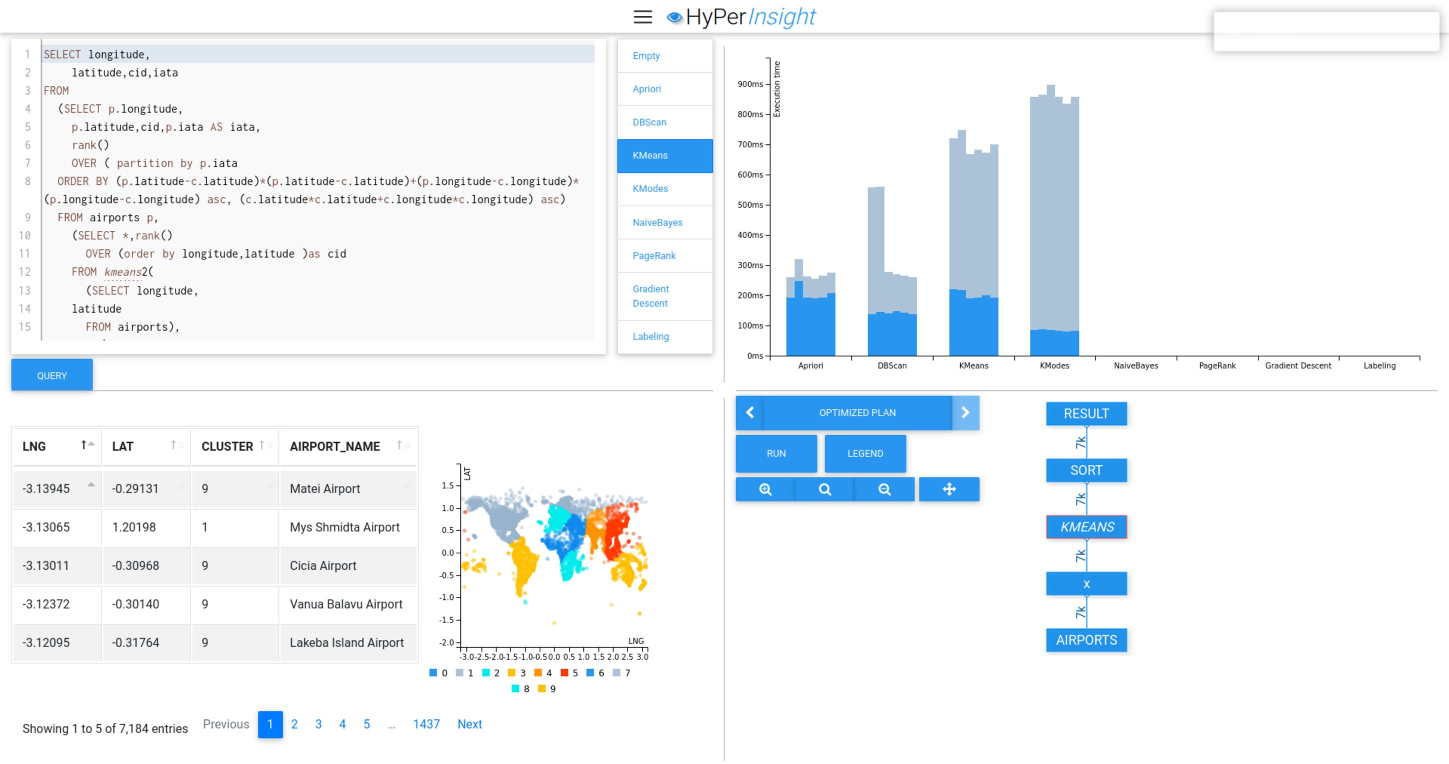
PageRank

ID	PRE	POST	SUPPORT	CONFIDENCE
1	{MUC}	{ZRH}	0.10219	0.72727
3	{ZRH,MUC}	{FRA}	0.09854	0.96429
5	{MUC}	{ZRH,FRA}	0.09854	0.70130
6	{MUC,FRA}	{ZRH}	0.09854	0.80597
31	{MUC}	{CDG}	0.10219	0.72727

Association Rules

Apriori

k-Means


HyPerInsight
SELECT longitude,
latitude,cid,iata
FROM
(SELECT p.longitude,
p.latitude,cid,p.iata AS iata,
rank()
OVER (partition by p.iata
ORDER BY (p.latitude-c.latitude)*(p.latitude-c.latitude)+(p.longitude-c.longitude)*
(p.longitude-c.longitude) asc, (c.latitude*c.latitude+c.longitude*c.longitude) asc)
FROM airports p,
(SELECT *,rank()
OVER (order by longitude,latitude)as cid
FROM kmeans2(
(SELECT longitude,
latitude
FROM airports),
QUERY
Empty
Apriori
DBScan
KMeans
KModes
NaiveBayes
PageRank
Gradient Descent
Labeling
Execution time
900ms
800ms
700ms
600ms
500ms
400ms
300ms
200ms
100ms
0ms
Apriori
DBScan
KMeans
KModes
NaiveBayes
PageRank
Gradient Descent
Labeling
LNG
LAT
CLUSTER
AIRPORT_NAME
-3.13945
-0.29131
9
Matei Airport
-3.13065
1.20198
1
Mys Shmidta Airport
-3.13011
-0.30968
9
Cicia Airport
-3.12372
-0.30140
9
Vanua Balavu Airport
-3.12095
-0.31764
9
Lakeba Island Airport
Showing 1 to 5 of 7,184 entries
Previous
1
2
3
4
5
...
1437
Next
LAT
LNG
0 1 2 3 4 5 6 7
8 9
OPTIMIZED PLAN
RUN
LEGEND
RESULT
7k
SORT
7k
KMEANS
7k
x
7k
AIRPORTS

k-Means

- n-dimensional points $x \in P \subset \mathbf{R}^n$
- k clusters C with k points forming the initial centres $C_0 \subset P, |C_0| = k$.
- A point belongs to the closest located cluster $c \in C$ based on a metric like Euclidean distance ($||c - x||_2$)
- Eq. (1) returns all points of a cluster.
- new centre: average of all points (2):

$$cluster(c) = \{x | x \in P : \nexists d \in C : d \neq c \land ||d - x||_2 < ||c - x||_2\}, \tag{1}$$

$$c_{i+1} = \sum_{x \in cluster(c_i)} \frac{x}{|cluster(c_i)|}. \tag{2}$$

- k tuples: centres (line 2)
- each iteration: update coordinates (line 4-10).
- window function: ranking of closest centres per point (line 5-86).

```
1  with recursive clusters (iter, cid, x, y) as (
2    (select 0,id, x, y from points limit 5)
3  union all
4    select iter+1,cid, avg(px), avg(py) from (
5      select iter, cid, p.x as px, p.y as py, rank()
6        over (partition by p.id order by
7          (p.x-c.x)*(p.x-c.x)+(p.y-c.y)*(p.y-c.y) asc,
8          (c.x*c.x+c.y*c.y) asc)
9      from points p, clusters c) x
10   where x.rank=1 and iter<100 group by cid, iter
11 ) select * from clusters where iter=100;
```

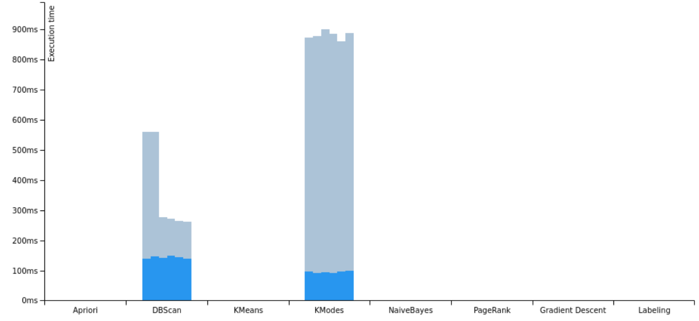
DBSCAN

HyPer*Insight*

```sql
SELECT cl.longitude,
    cl.latitude,
    cluster,
    a.airport_name
FROM
  (SELECT *
  FROM dbscan(
    (SELECT longitude,
    latitude
    FROM airports
    WHERE longitude
      BETWEEN -20
      AND 20),3,4)) cl, airports a
  WHERE cl.longitude=a.longitude
  AND cl.latitude=a.latitude;
```

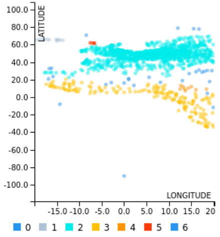
QUERY

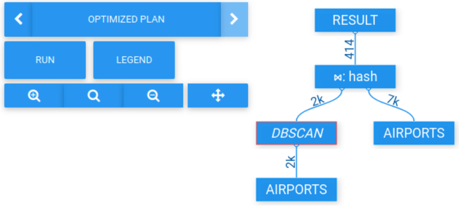
- Empty
- Apriori
- DBScan
- KMeans
- KModes
- NaiveBayes
- PageRank
- Gradient Descent
- Labeling



LONGITUDE	LATITUDE	CLUSTER	AIRPORT_NAME
-19.57280	65.73170	1	Sauðárkrókur Airport
-18.91670	66.13330	1	Siglufjörður Airport
-18.07270	65.66000	1	Akureyri Airport
-18.01730	66.54580	1	Grímsey Airport
-17.88710	27.81480	2	Hierro Airport

Showing 1 to 5 of 1,455 entries Previous 1 2 3 4 5 ... 291 Next





DBSCAN

- ε: maximal distance between two points
- minimal number of points per cluster *minPoints* > 1, declared as noise otherwise.

For every point within a cluster $x \in G \subset P \subset \mathbf{R}^n$, another point $y \in G$ exists, whose distance to x is less than ε:

$$\forall x \in G : \exists y \in G : x \neq y \wedge ||x - y||_2 < \varepsilon. \tag{3}$$

- First, each point forms its own cluster (line 2).
- clusters that are less than ε away are merged (line 4-8).

```
1  with recursive dbscan(iter,id,x,y,clusterid,noise)
2  as ( select 0,id,x,y,id,true from points
3  union all
4    select iter+1,p.id,p.x,p.y, min(c.clusterid),
5           count(*) < 3
6    from points p, dbscan c
7    where iter<10 and (p.x-c.x)^2+(p.y-c.y)^2<1.5^2
8    group by iter,p.id,p.x,p.y
9  )
10 select * from dbscan where iter=10;
```

PageRank


HyPerInsight
SELECT row_number()
OVER (order by "rank" desc), "rank", airport_name
FROM airports a,
(SELECT *
FROM pagerank(
(SELECT source_airport,
destination_airport
FROM routes), λ(src) (src.source_airport), λ(dst) (dst.destination_airport))) pg
WHERE a.iata = pg.node
QUERY
Empty
Apriori
DBScan
KMeans
KModes
NaiveBayes
PageRank
Gradient Descent
Labeling
Execution time
900ms
800ms
700ms
600ms
500ms
400ms
300ms
200ms
100ms
0ms
Apriori
DBScan
KMeans
KModes
NaiveBayes
PageRank
Gradient Descent
Labeling
ROW_NUMBER
RANK
AIRPORT_NAME
1
0.00923
Hartsfield Jackson...
2
0.00581
Chicago O'Hare...
3
0.00561
Los Angeles...
4
0.00533
Dallas Fort Worth...
5
0.00490
Charles de Gaulle...
ROW_NUMBER
0
500
1000
1500
2000
2500
3000
RANK
Showing 1 to 5 of 3,186 entries
Previous
1
2
3
4
5
...
638
Next
OPTIMIZED PLAN
RUN
LEGEND
RESULT
257k
WINDOW
257k
⋈: hash
7k
AIRPORTS
PAGERANK
68k
σ
68k
ROUTES

PageRank

- importance of a web pages
- a link directing to another web page: an edge $(s, d) \in N \times N$.

1. each node: same PageRank value pr_0 (4).
2. per iteration: each node s distributes its own value $pr_i(s)$ equally to all outgoing edges $(s, d) \in E$.
3. new PageRank value $pr_{i+1}(n)$ of a node n: sum of the values of all incoming edges $(s, n) \in E$, damping factor α (5):

$$pr_0(n) := \frac{1}{|N|}, \tag{4}$$

$$pr_{i+1}(n) := \alpha \cdot \sum_{(s,n)\in E} \frac{pr_i(s)}{|\{d \mid (s,d) \in E\}|} + \frac{1-\alpha}{|N|}. \tag{5}$$

1. base case (line 2) computes the initial PageRank value pr_0.
2. recursive step: divides each node's PageRank value by the number of outgoing edges (line 11-14), assigns and sums up this fraction for each destination node `dst` (line 9-18).

```
1  with recursive pagerank (iter,node,pr) as (
2   select 0, e.dst, 1::float/(
3      select count(distinct dst) from edges)
4   from edges e
5   group by e.dst
6  union all
7   select iter+1,dst,0.1*((1::float/
8     (select count(distinct dst)
9   from edges))+0.9*sum(b)
10  from (
11   select iter, e.dst, p.pr/(
12     select count (*)
13     from edges x
14     where x.src=e.src) as b
15   from edges e, pagerank p
16   where e.src=p.Node and iter < 100
17  ) i
18  group by dst, iter
19 ) select * from pagerank where iter=100;
```

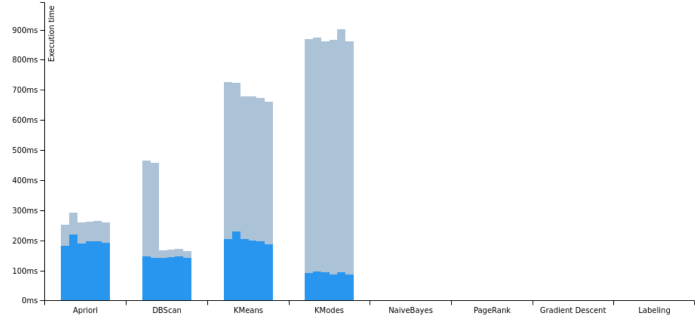
Apriori

HyPer*Insight*

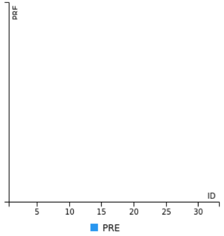
```sql
with airline_rules AS
  (SELECT *
  FROM apriori(
    (SELECT airline_id,
    destination_airport
    FROM routes), 0.1))
  SELECT *
FROM airline_rules
WHERE 'MUC' = ANY(airline_rules."Pre");
```

- Empty
- Apriori
- DBScan
- KMeans
- KModes
- NaiveBayes
- PageRank
- Gradient Descent
- Labeling

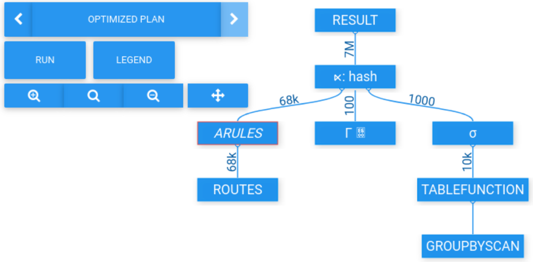
QUERY



ID	PRE	POST	SUPPORT	CONFIDENCE
1	{MUC}	{ZRH}	0.10219	0.72727
3	{ZRH,MUC}	{FRA}	0.09854	0.96429
5	{MUC}	{ZRH,FRA}	0.09854	0.70130
6	{MUC,FRA}	{ZRH}	0.09854	0.80597
31	{MUC}	{CDG}	0.10219	0.72727



Showing 1 to 5 of 6 entries Previous 1 2 Next



Apriori

- Determining the frequent item sets for the Apriori algorithm
- a recursively growing relation calculates the frequent item sets: starting with the one-element item sets (each item as an array with one element).

```
with recursive transactions (tid, bucket) as (
--one array per shopping cart
 select tid, array_agg(item) from sales group by tid
-- frequent item sets of size 1
),sales_supp as (
 select item from sales group by item having count(*)>=10
),frequentitemsets as ( -- frequent item sets with support >= 10
  -- with one element
 (select distinct array[p.item]::int[] as items from sales_supp p)
 union all ( -- extend item sets recursively by one element
   select distinct array_append(t.items,p.item::int)::int[]
   from frequentitemsets t, sales_supp p
   where 10 <= ( -- count support
      select count(*) from transactions t2
      where array_append(t.items,p.item::int)::int[] <@ ( t2.bucket )
   ) and t.items[(select count(*) from unnest(t.items))]<p.item
))
select * from frequentitemsets;
```

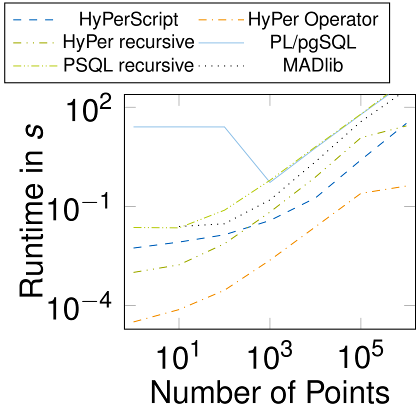
Evaluation

Evaluation

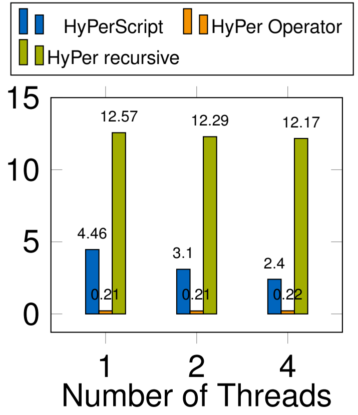
- *HyPerScript*, table operators of HyPer and MADlib, recursive tables in HyPer, PostgreSQL and Umbra, and *PL/pgSQL* procedures (PostgreSQL 12.6 with MADlib 1.17.0)
- Ubuntu 20.04 LTS machine with six Intel Core i7-3930K CPUs running at 3.20 GHz and 64 GB DDR4 RAM

Evaluation: k-Means

- 10^6 points, x- and as y-coordinates equally distributed in $[0, 10^6]$
- runtime grows linearly with the input size
- integrated operators the best
- implementation within *HyPerScript*/recursive table comparable performance, outperform PostgreSQL.
- *HyPerScript*: 30 % faster with each additional core



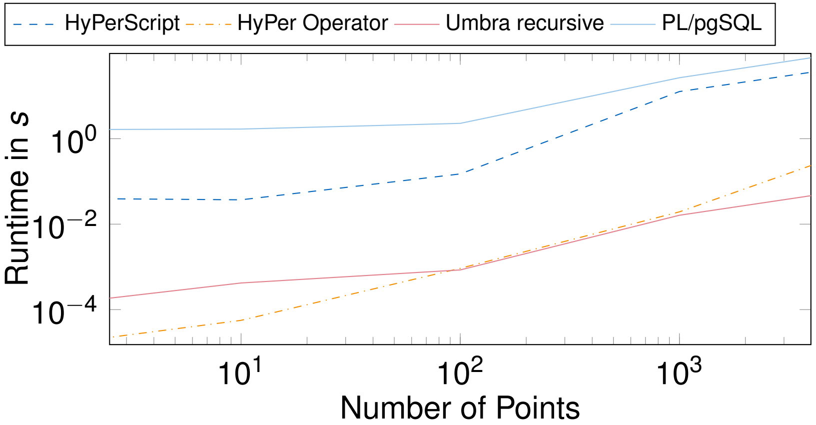
Input size



Scalability

Evaluation: DBSCAN

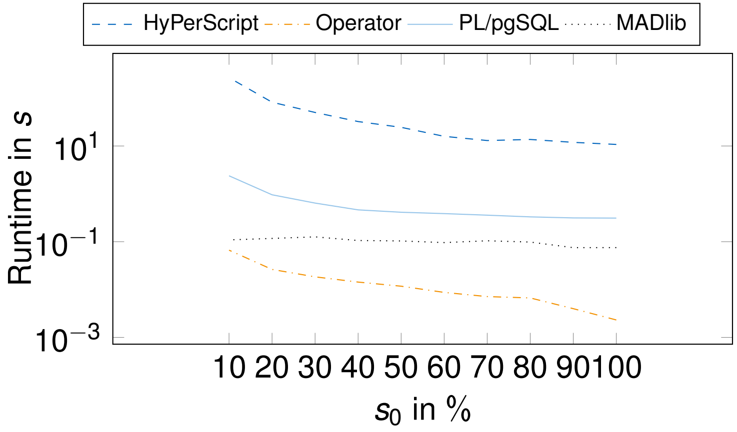
- 10^6 points, x- and as y-coordinates equally distributed in $[0, 10^6]$
- recursive computation in Umbra as fast as the implemented operator



$\varepsilon = 20$, $\mathtt{minPts} = 2$, 100 iterations, 12 threads.

Evaluation: Apriori

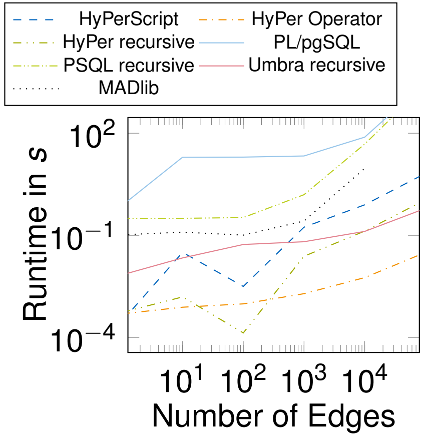
- 100 different items and 1000 shopping carts synthetically generated.
- The number of items per shopping cart: between 0 and 10.
- With increasing minimum support, the runtime decreases as less frequent item sets exist.
- larger s_0, the lower the number of frequent item sets and thus the lower the runtime.
- HyPer operator performs the best,
- implementation in *HyPerScript* slower than *PL/pgSQL* (cause: implementation of the array set operator that unnests the array).



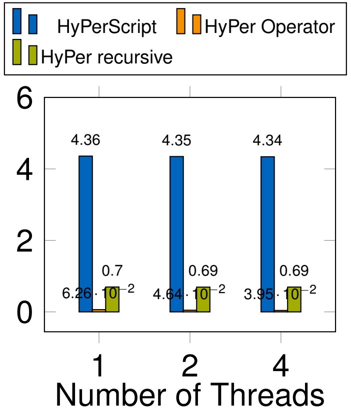
12 threads, minimum support s_0

Evaluation: PageRank

- The PageRank value was computed for 10^5 nodes with the same number of edges.
- implementations in HyPer outperform counterparts in PostgreSQL
- scripting language: slightly worse than using a recursive table in PostgreSQL and HyPer
- few edges: additional overhead of the integrated operator in HyPer (dictionary for the nodes and edges in a sparse matrix as Compressed-Sparse-Row (CSR))
- With an increasing number of edges, additional effort amortised, so operator faster than the script function



Number of edges



Scalability

Conclusion

- data mining algorithms in SQL-92 by relying on recursive CTEs
- four algorithms: k-Means, DBSCAN, Apriori and PageRank in SQL
- evaluation: worse than the operators in HyPer but similar to MADlib's library functions
- within recursive tables: support of aggregate and window functions necessary

Thank you for your attention!