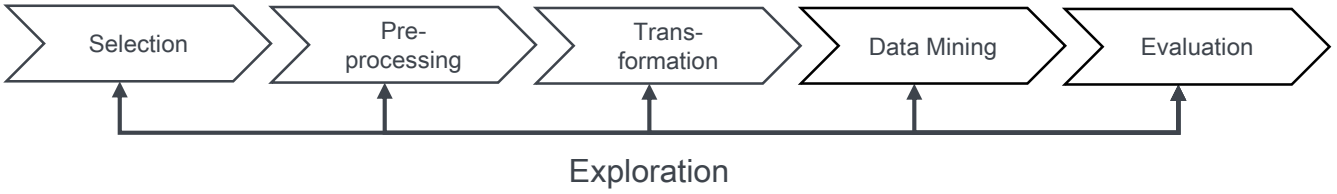
University of Stuttgart
Germany

Quality-Driven Early Stopping for Explorative Cluster Analysis for Big Data

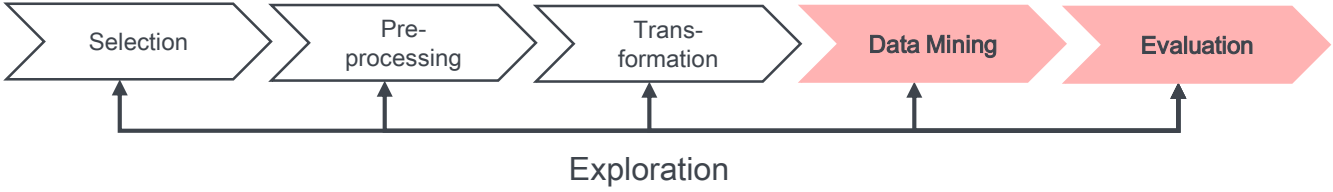
Manuel Fritz, Michael Behringer, Holger Schwarz

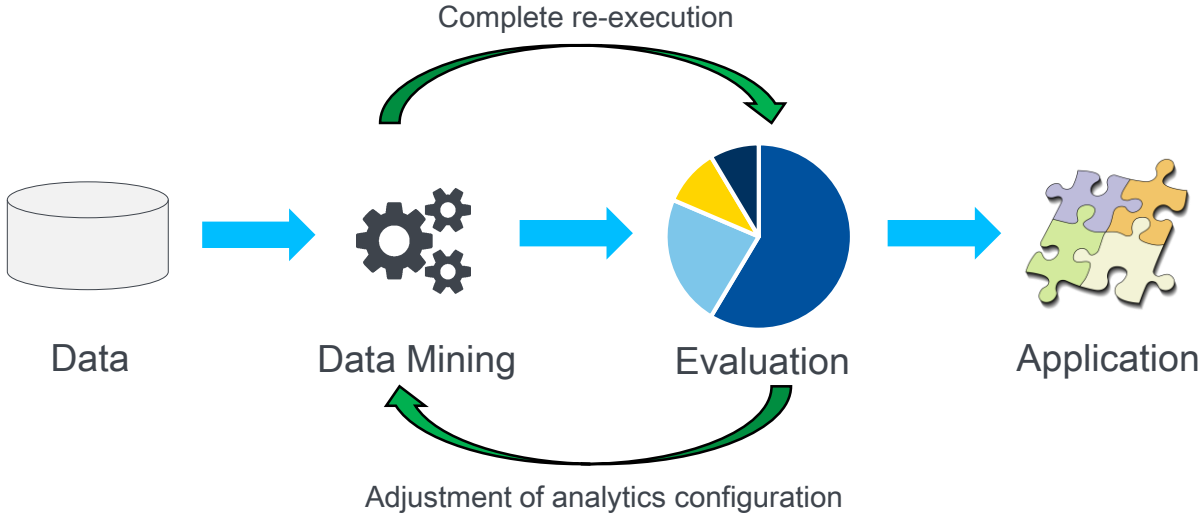
IPVS
University of Stuttgart

Motivation

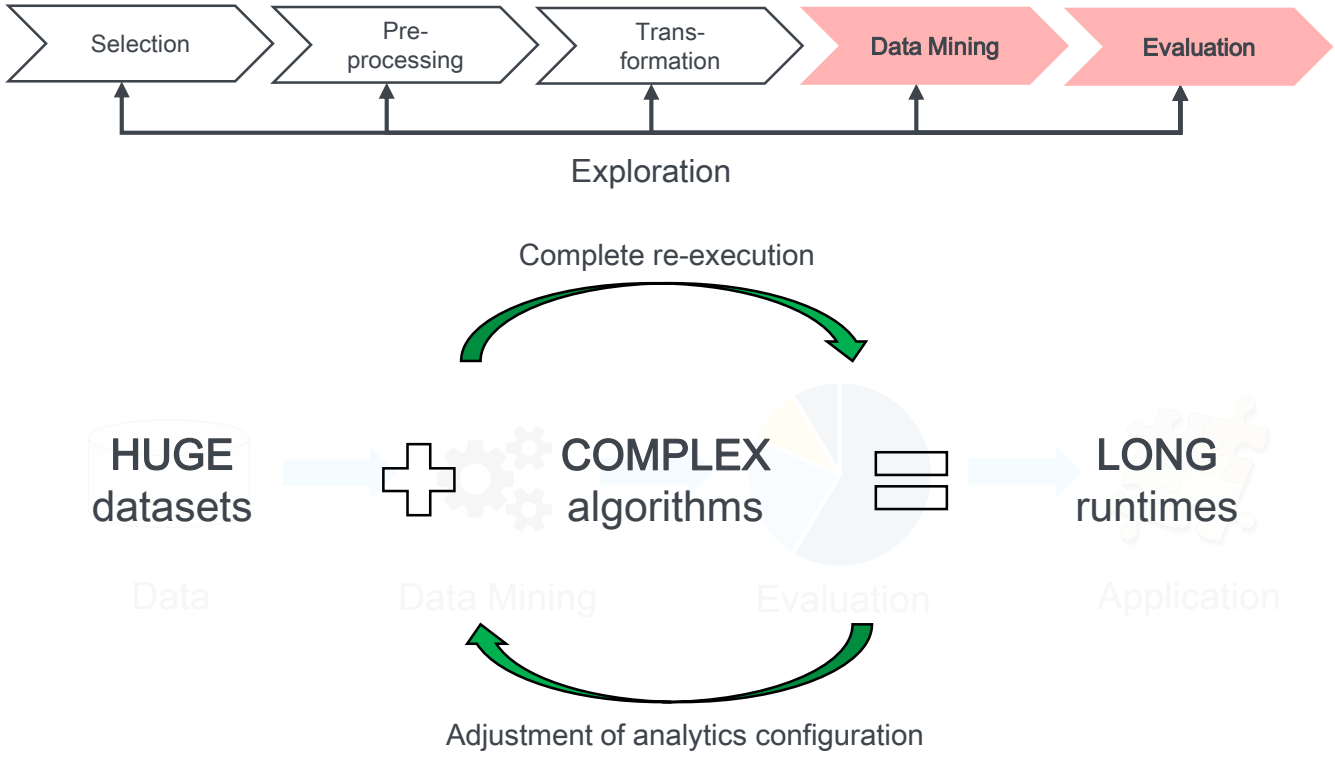

Selection
Pre-
processing
Trans-
formation
Data Mining
Evaluation
Exploration

Motivation


Selection
Pre-
processing
Trans-
formation
Data Mining
Evaluation
Exploration


Complete re-execution
Data
Data Mining
Evaluation
Application
Adjustment of analytics configuration

Motivation

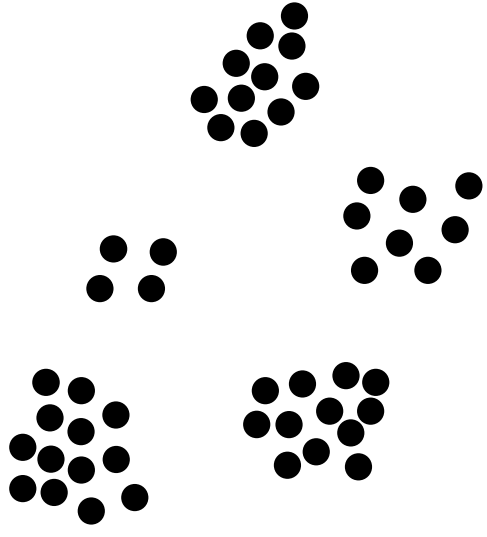

Selection
Pre-processing
Trans-formation
Data Mining
Evaluation
Exploration
Complete re-execution
HUGE datasets
COMPLEX algorithms
LONG runtimes
Data
Data Mining
Evaluation
Application
Adjustment of analytics configuration

Partitioning Clustering Algorithms

Algorithm 1: Skeleton for partitioning clustering algorithms

Input : $D = \{d_1, d_2, ..., d_n\}$ (set of n entities)
k (number of clusters)
Output: $C = \{c_1, c_2, ..., c_k\}$ (set of k centroids)
$m(d_i) \mid 1 \leq i \leq n$ (representation function mapping entity d_i to the closest cluster)

```
1  foreach c_i ∈ C do
       /* Seed initial centroids                    */
2      initialize(c_i);
3  end
4  foreach d_i ∈ D do
       /* m(d_i) holds the membership of            */
       /* entity d_i to its cluster                 */
5      m(d_i) = closestCluster(d_i);
6  end
7  repeat
8      foreach c_i ∈ C do
           /* Calculate new position                */
           /* of all centroids                      */
9          updateCentroid(c_i);
10     end
11     foreach d_i ∈ D do
           /* Reassign entities to centroids        */
12         closest = closestCluster(d_i);
13         if closest ≠ m(d_i) then
14             m(d_i) = closest;
15         end
16     end
17 until convergenceReached();
```

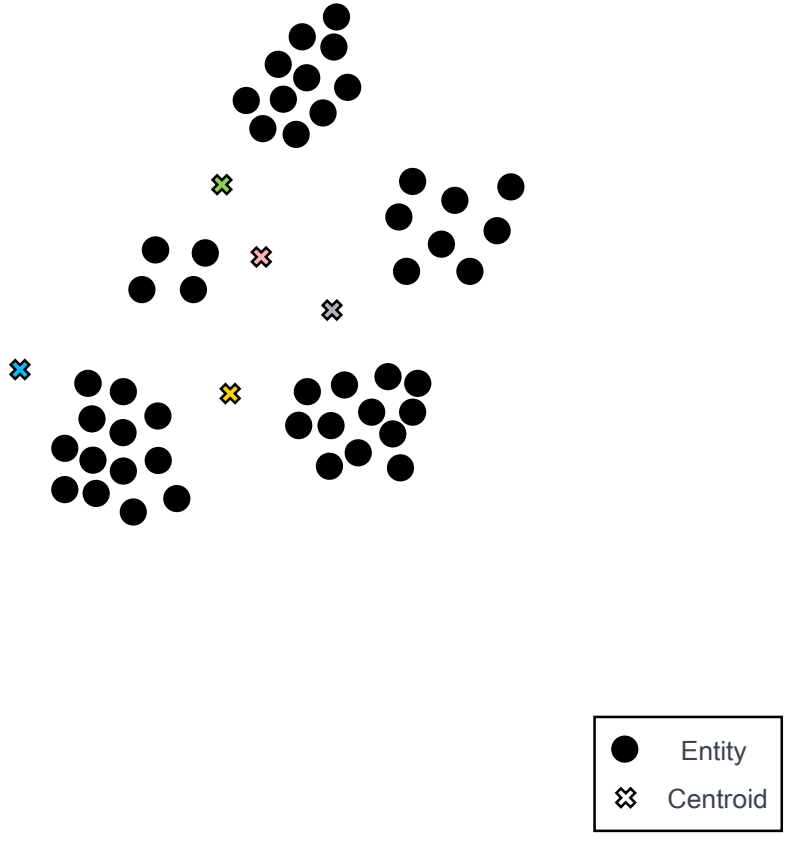


Partitioning Clustering Algorithms

Algorithm 1: Skeleton for partitioning clustering algorithms

Input : $D = \{d_1, d_2, ..., d_n\}$ (set of n entities)
k (number of clusters)
Output: $C = \{c_1, c_2, ..., c_k\}$ (set of k centroids)
$m(d_i) \mid 1 \leq i \leq n$ (representation function mapping entity d_i to the closest cluster)

```
1  foreach c_i ∈ C do
       /* Seed initial centroids                    */
2      initialize(c_i);
3  end
4  foreach d_i ∈ D do
       /* m(d_i) holds the membership of            */
       /* entity d_i to its cluster                 */
5      m(d_i) = closestCluster(d_i);
6  end
7  repeat
8      foreach c_i ∈ C do
           /* Calculate new position                */
           /* of all centroids                      */
9          updateCentroid(c_i);
10     end
11     foreach d_i ∈ D do
           /* Reassign entities to centroids        */
12         closest = closestCluster(d_i);
13         if closest ≠ m(d_i) then
14             m(d_i) = closest;
15         end
16     end
17 until convergenceReached();
```

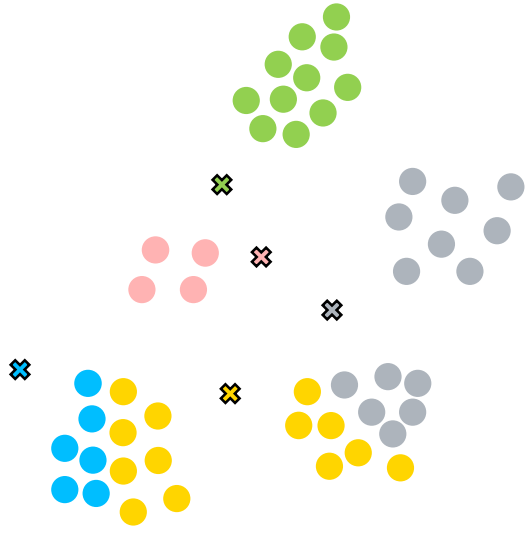


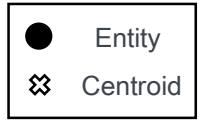
Partitioning Clustering Algorithms

Algorithm 1: Skeleton for partitioning clustering algorithms

Input : $D = \{d_1, d_2, ..., d_n\}$ (set of n entities)
k (number of clusters)
Output: $C = \{c_1, c_2, ..., c_k\}$ (set of k centroids)
$m(d_i) \mid 1 \leq i \leq n$ (representation function mapping entity d_i to the closest cluster)

```
1  foreach c_i ∈ C do
       /* Seed initial centroids                    */
2      initialize(c_i);
3  end
4  foreach d_i ∈ D do
       /* m(d_i) holds the membership of            */
       /* entity d_i to its cluster                 */
5      m(d_i) = closestCluster(d_i);
6  end
7  repeat
8      foreach c_i ∈ C do
           /* Calculate new position                */
           /* of all centroids                      */
9          updateCentroid(c_i);
10     end
11     foreach d_i ∈ D do
           /* Reassign entities to centroids        */
12         closest = closestCluster(d_i);
13         if closest ≠ m(d_i) then
14             m(d_i) = closest;
15         end
16     end
17 until convergenceReached();
```



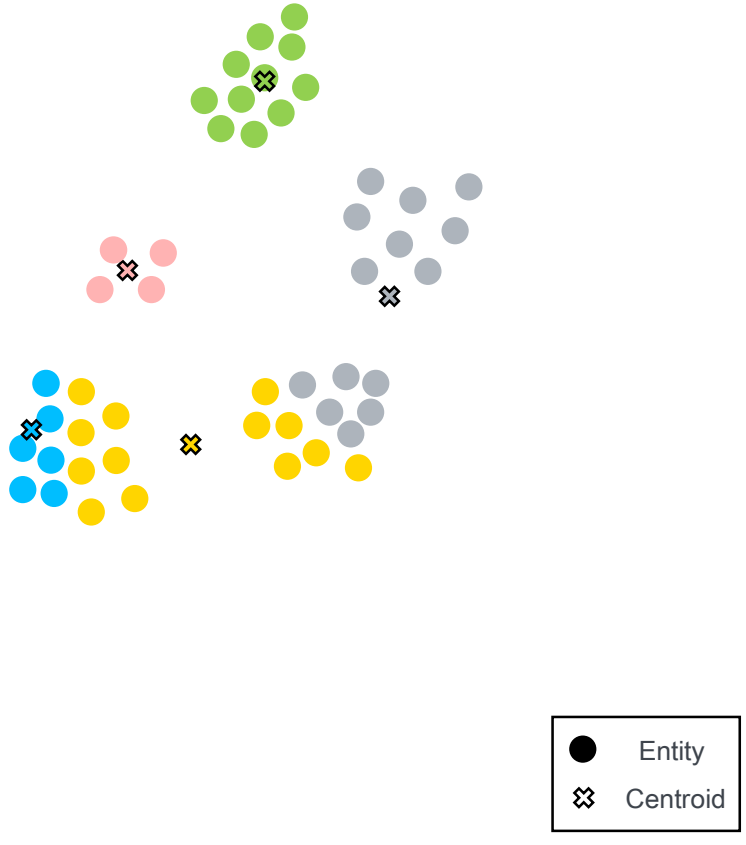


Partitioning Clustering Algorithms

Algorithm 1: Skeleton for partitioning clustering algorithms

Input : $D = \{d_1, d_2, ..., d_n\}$ (set of n entities)
k (number of clusters)
Output: $C = \{c_1, c_2, ..., c_k\}$ (set of k centroids)
$m(d_i) \mid 1 \leq i \leq n$ (representation function mapping entity d_i to the closest cluster)

```
1  foreach c_i ∈ C do
       /* Seed initial centroids                    */
2      initialize(c_i);
3  end
4  foreach d_i ∈ D do
       /* m(d_i) holds the membership of            */
       /* entity d_i to its cluster                 */
5      m(d_i) = closestCluster(d_i);
6  end
7  repeat
8      foreach c_i ∈ C do
           /* Calculate new position                */
           /* of all centroids                      */
9          updateCentroid(c_i);
10     end
11     foreach d_i ∈ D do
           /* Reassign entities to centroids        */
12         closest = closestCluster(d_i);
13         if closest ≠ m(d_i) then
14             m(d_i) = closest;
15         end
16     end
17 until convergenceReached();
```

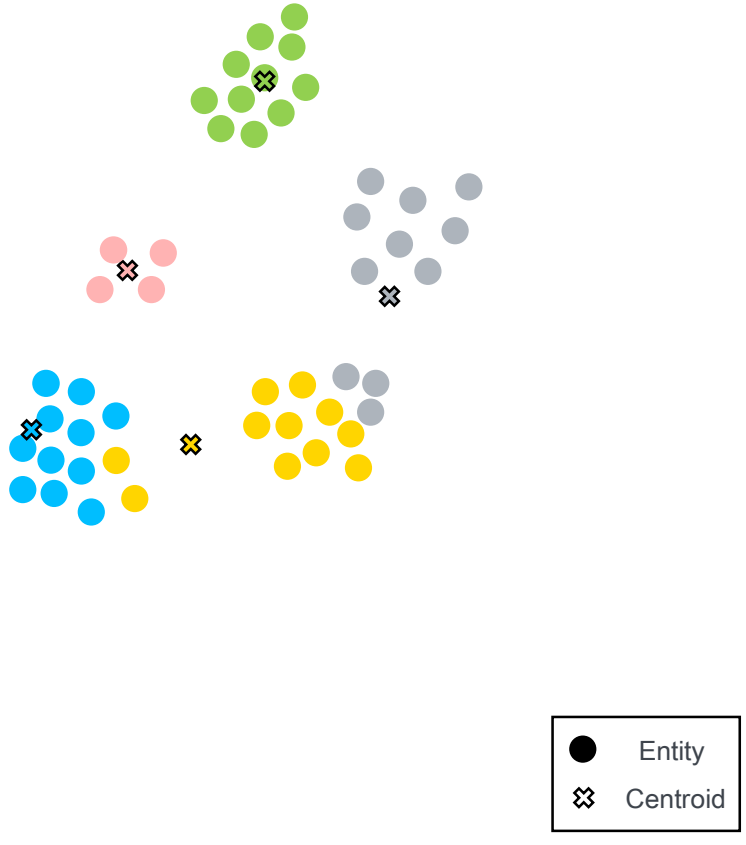


Partitioning Clustering Algorithms

Algorithm 1: Skeleton for partitioning clustering algorithms

Input : $D = \{d_1, d_2, ..., d_n\}$ (set of n entities)
k (number of clusters)
Output: $C = \{c_1, c_2, ..., c_k\}$ (set of k centroids)
$m(d_i) \mid 1 \leq i \leq n$ (representation function mapping entity d_i to the closest cluster)

```
1  foreach c_i ∈ C do
       /* Seed initial centroids                    */
2      initialize(c_i);
3  end
4  foreach d_i ∈ D do
       /* m(d_i) holds the membership of            */
       /* entity d_i to its cluster                 */
5      m(d_i) = closestCluster(d_i);
6  end
7  repeat
8      foreach c_i ∈ C do
           /* Calculate new position                */
           /* of all centroids                      */
9          updateCentroid(c_i);
10     end
11     foreach d_i ∈ D do
           /* Reassign entities to centroids        */
12         closest = closestCluster(d_i);
13         if closest ≠ m(d_i) then
14             m(d_i) = closest;
15         end
16     end
17 until convergenceReached();
```

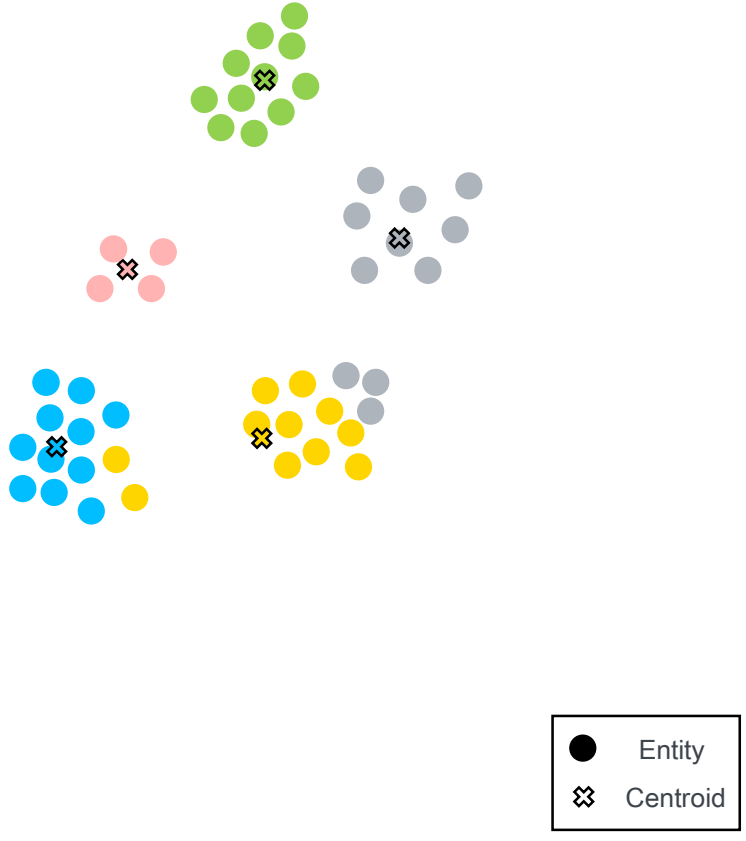


Partitioning Clustering Algorithms

Algorithm 1: Skeleton for partitioning clustering algorithms

Input : $D = \{d_1, d_2, ..., d_n\}$ (set of n entities)
k (number of clusters)
Output: $C = \{c_1, c_2, ..., c_k\}$ (set of k centroids)
$m(d_i) \mid 1 \leq i \leq n$ (representation function mapping entity d_i to the closest cluster)

```
1  foreach c_i ∈ C do
       /* Seed initial centroids                    */
2      initialize(c_i);
3  end
4  foreach d_i ∈ D do
       /* m(d_i) holds the membership of            */
       /* entity d_i to its cluster                 */
5      m(d_i) = closestCluster(d_i);
6  end
7  repeat
8      foreach c_i ∈ C do
           /* Calculate new position                */
           /* of all centroids                      */
9          updateCentroid(c_i);
10     end
11     foreach d_i ∈ D do
           /* Reassign entities to centroids        */
12         closest = closestCluster(d_i);
13         if closest ≠ m(d_i) then
14             m(d_i) = closest;
15         end
16     end
17 until convergenceReached();
```

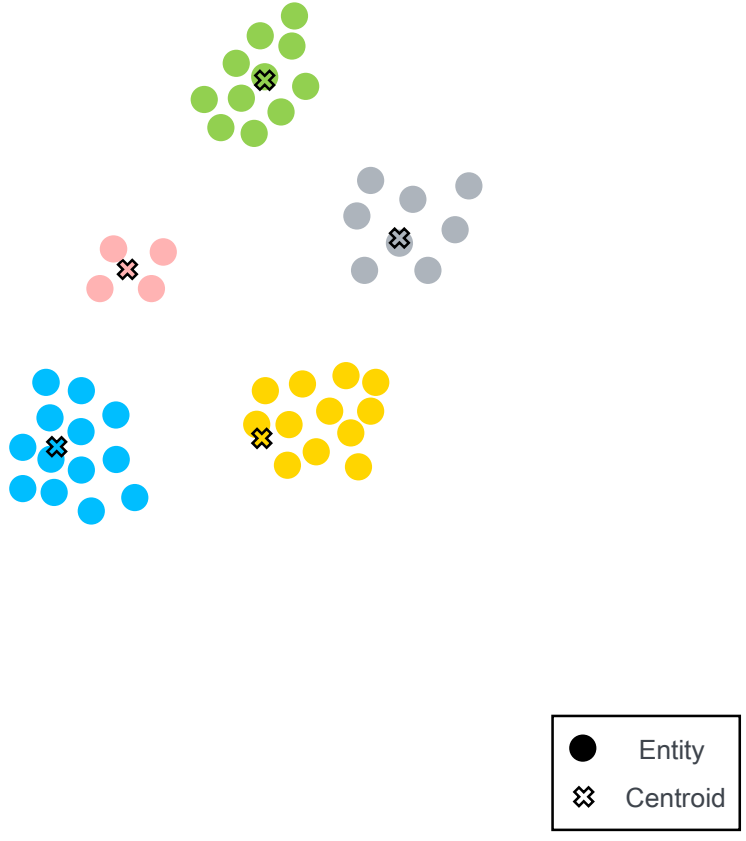


Partitioning Clustering Algorithms

Algorithm 1: Skeleton for partitioning clustering algorithms

Input : $D = \{d_1, d_2, ..., d_n\}$ (set of n entities)
k (number of clusters)
Output: $C = \{c_1, c_2, ..., c_k\}$ (set of k centroids)
$m(d_i) \mid 1 \leq i \leq n$ (representation function mapping entity d_i to the closest cluster)

```
1  foreach c_i ∈ C do
       /* Seed initial centroids                    */
2      initialize(c_i);
3  end
4  foreach d_i ∈ D do
       /* m(d_i) holds the membership of            */
       /* entity d_i to its cluster                 */
5      m(d_i) = closestCluster(d_i);
6  end
7  repeat
8      foreach c_i ∈ C do
           /* Calculate new position                */
           /* of all centroids                      */
9          updateCentroid(c_i);
10     end
11     foreach d_i ∈ D do
           /* Reassign entities to centroids        */
12         closest = closestCluster(d_i);
13         if closest ≠ m(d_i) then
14             m(d_i) = closest;
15         end
16     end
17 until convergenceReached();
```

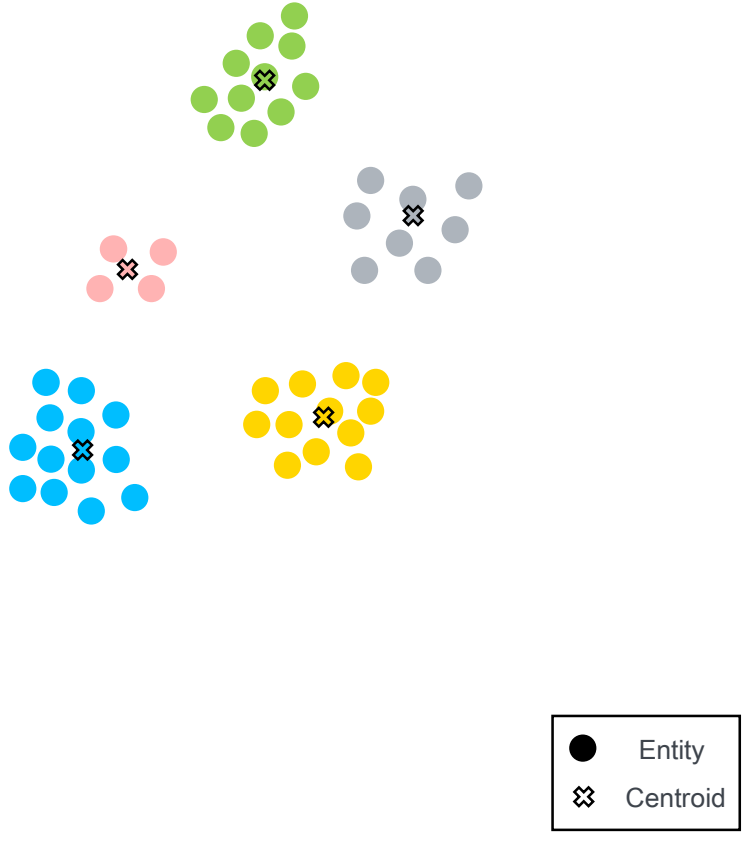


Partitioning Clustering Algorithms

Algorithm 1: Skeleton for partitioning clustering algorithms

Input : $D = \{d_1, d_2, ..., d_n\}$ (set of n entities)
k (number of clusters)
Output: $C = \{c_1, c_2, ..., c_k\}$ (set of k centroids)
$m(d_i) \mid 1 \leq i \leq n$ (representation function mapping entity d_i to the closest cluster)

```
1  foreach c_i ∈ C do
       /* Seed initial centroids                    */
2      initialize(c_i);
3  end
4  foreach d_i ∈ D do
       /* m(d_i) holds the membership of            */
       /* entity d_i to its cluster                 */
5      m(d_i) = closestCluster(d_i);
6  end
7  repeat
8      foreach c_i ∈ C do
           /* Calculate new position                */
           /* of all centroids                      */
9          updateCentroid(c_i);
10     end
11     foreach d_i ∈ D do
           /* Reassign entities to centroids        */
12         closest = closestCluster(d_i);
13         if closest ≠ m(d_i) then
14             m(d_i) = closest;
15         end
16     end
17 until convergenceReached();
```

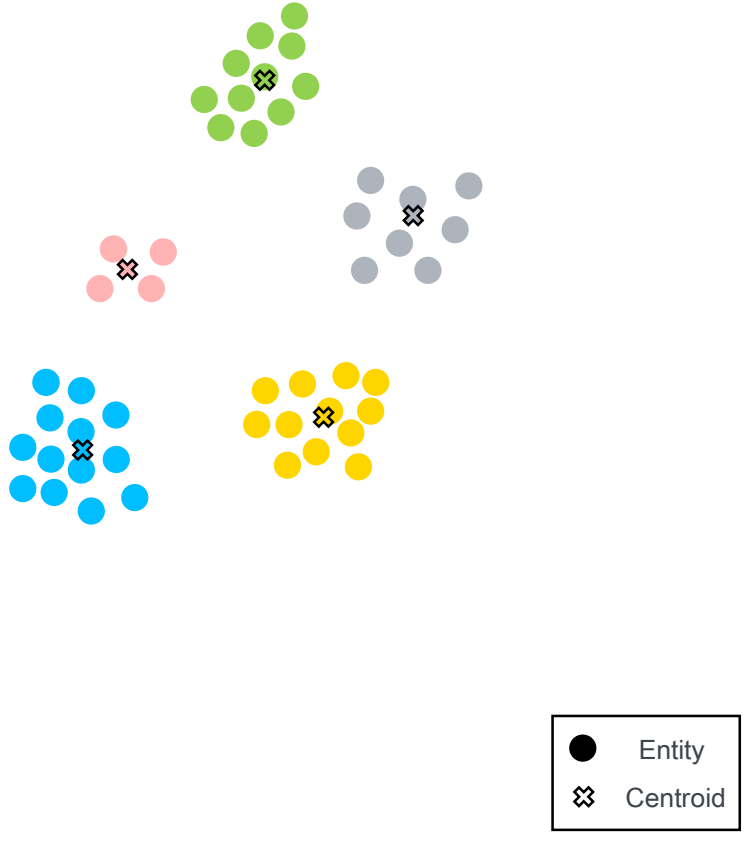


Partitioning Clustering Algorithms

Algorithm 1: Skeleton for partitioning clustering algorithms

Input : $D = \{d_1, d_2, ..., d_n\}$ (set of n entities)
k (number of clusters)
Output: $C = \{c_1, c_2, ..., c_k\}$ (set of k centroids)
$m(d_i) \mid 1 \leq i \leq n$ (representation function mapping entity d_i to the closest cluster)

```
1  foreach c_i ∈ C do
       /* Seed initial centroids                    */
2      initialize(c_i);
3  end
4  foreach d_i ∈ D do
       /* m(d_i) holds the membership of            */
       /* entity d_i to its cluster                 */
5      m(d_i) = closestCluster(d_i);
6  end
7  repeat
8      foreach c_i ∈ C do
           /* Calculate new position                */
           /* of all centroids                      */
9          updateCentroid(c_i);
10     end
11     foreach d_i ∈ D do
           /* Reassign entities to centroids        */
12         closest = closestCluster(d_i);
13         if closest ≠ m(d_i) then
14             m(d_i) = closest;
15         end
16     end
17 until convergenceReached();
```

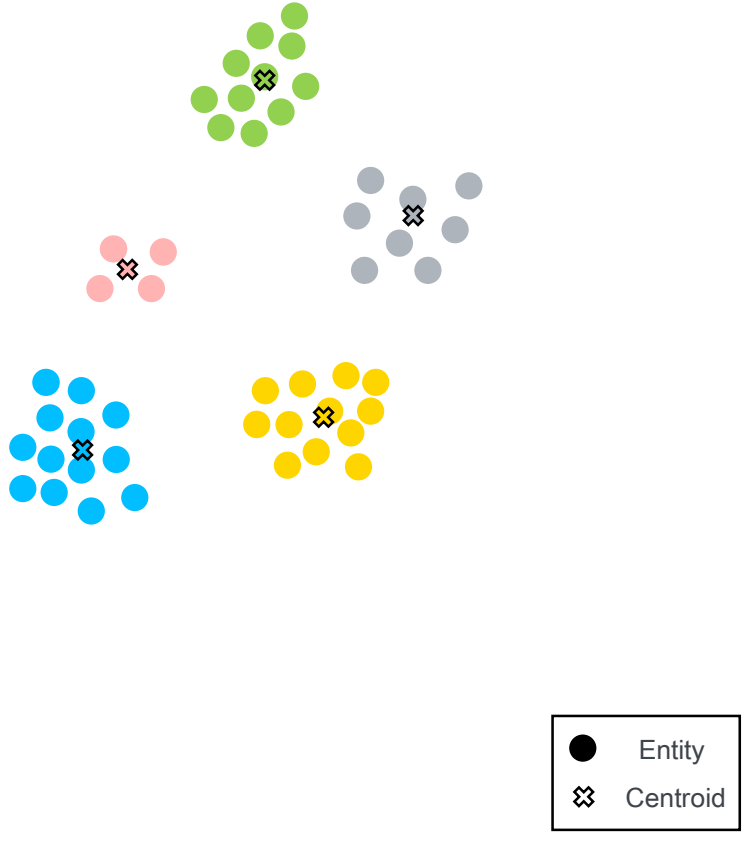


Partitioning Clustering Algorithms

Algorithm 1: Skeleton for partitioning clustering algorithms

Input : $D = \{d_1, d_2, ..., d_n\}$ (set of n entities)
k (number of clusters)
Output: $C = \{c_1, c_2, ..., c_k\}$ (set of k centroids)
$m(d_i) \mid 1 \leq i \leq n$ (representation function mapping entity d_i to the closest cluster)

```
1  foreach c_i ∈ C do
       /* Seed initial centroids                    */
2      initialize(c_i);
3  end
4  foreach d_i ∈ D do
       /* m(d_i) holds the membership of            */
       /* entity d_i to its cluster                 */
5      m(d_i) = closestCluster(d_i);
6  end
7  repeat
8      foreach c_i ∈ C do
           /* Calculate new position                */
           /* of all centroids                      */
9          updateCentroid(c_i);
10     end
11     foreach d_i ∈ D do
           /* Reassign entities to centroids        */
12         closest = closestCluster(d_i);
13         if closest ≠ m(d_i) then
14             m(d_i) = closest;
15         end
16     end
17 until convergenceReached();
```

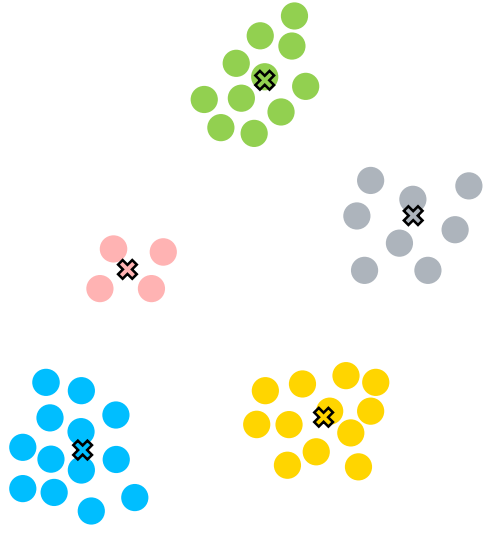


Partitioning Clustering Algorithms

Algorithm 1: Skeleton for partitioning clustering algorithms

Input : $D = \{d_1, d_2, ..., d_n\}$ (set of n entities)
k (number of clusters)
Output: $C = \{c_1, c_2, ..., c_k\}$ (set of k centroids)
$m(d_i) \mid 1 \leq i \leq n$ (representation function mapping entity d_i to the closest cluster)

```
1  foreach c_i ∈ C do
       /* Seed initial centroids                    */
2      initialize(c_i);
3  end
4  foreach d_i ∈ D do
       /* m(d_i) holds the membership of            */
       /* entity d_i to its cluster                 */
5      m(d_i) = closestCluster(d_i);
6  end
7  repeat
8      foreach c_i ∈ C do
           /* Calculate new position                */
           /* of all centroids                      */
9          updateCentroid(c_i);
10     end
11     foreach d_i ∈ D do
           /* Reassign entities to centroids        */
12         closest = closestCluster(d_i);
13         if closest ≠ m(d_i) then
14             m(d_i) = closest;
15         end
16     end
17 until convergenceReached();
```



Goal: Quality-Driven Early Stopping Criterion

Entity
Centroid

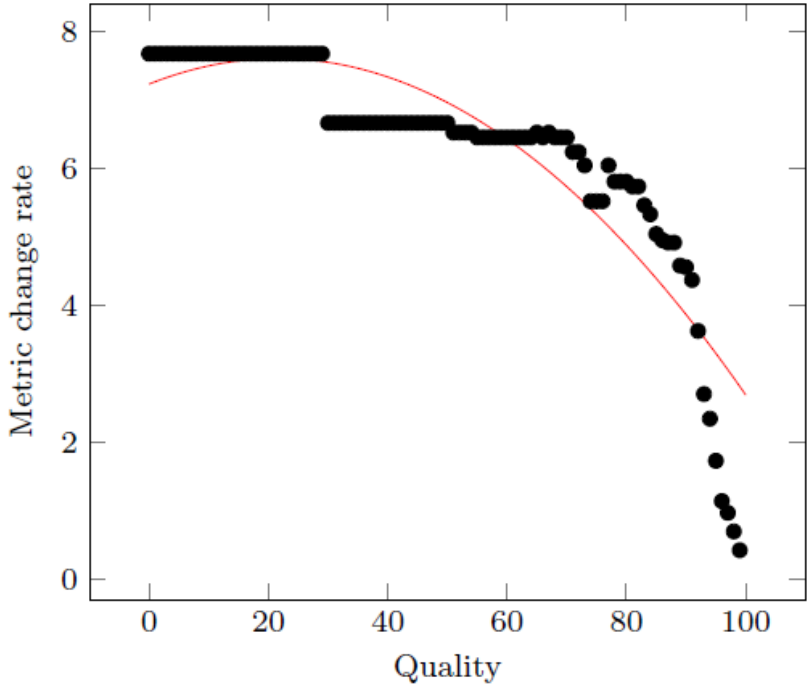
Metrics

Metric	Abbreviation	Formula	Normalized
Separation	SEP	$\sum_{i=1}^{k} \sum_{j=1}^{k} dist(z_i, z_j)$	✗
Compactness	COM	$\sum_{i=1}^{k} \sum_{d_m \in Z_i} dist(z_i, d_m)$	✗
Sum of squared errors	SSE	$\sum_{i=1}^{k} \sum_{d_m \in Z_i} dist(z_i, d_m)^2$	✗
Dunn Index	DUI	$\frac{\min(SEP(z_i, z_j))}{\max(COM(Z_i))} \mid \forall z_i, z_j \in Z, z_i \neq z_j$	✗
Coggins-Jain Index	CJI	$\min\left(\frac{SEP(z_i, z_j)}{COM(Z_i)}\right) \mid \forall z_i, z_j \in Z, z_i \neq z_j$	✗
Davies-Bouldin Index	DBI	$\frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} \left(\frac{\frac{1}{\lvert Z_i \rvert} COM(Z_i) + \frac{1}{\lvert Z_j \rvert} COM(Z_j)}{SEP(z_i, z_j)}\right)$	✗
Silhouette coefficient	SIC	$avg\left(\frac{b(d_m) - a(d_m)}{\max(a(d_m), b(d_m))} \mid \forall d_m \in D\right)$	✓

$b(d_m) = min_{z_i \neq z_j}\{avg\{dist(d_m, d_l) \mid d_m \neq d_l, d_l \in Z_j\}\}$
$a(d_m) = avg\{dist(d_m, d_l) \mid d_m \neq d_l; d_m, d_l \in Z_i\}$

Correlation Analysis

Metric value and Quality



Our Approach

Quality-Driven Early Stopping

10 Data sets

Training

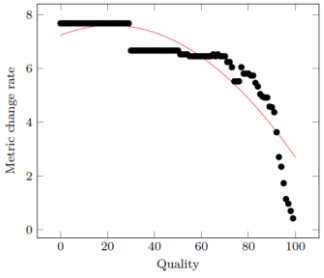
5 Data sets

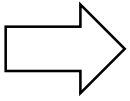
Application

5 Data sets

Algorithm 1: Skeleton for partitioning clustering algorithms

```
Input  : D = {d_1, d_2, ..., d_n} (set of n entities)
         k (number of clusters)
Output: C = {c_1, c_2, ..., c_k} (set of k centroids)
         m(d_i) | 1 ≤ i ≤ n (representation function
mapping entity d_i to the closest cluster)
1  foreach c_i ∈ C do
2      /* Seed initial centroids                    */
       initialize(c_i);
3  end
4  foreach d_i ∈ D do
       /* m(d_i) holds the membership of            */
       /* entity d_i to its cluster                 */
5      m(d_i) = closestCluster(d_i);
6  end
7  repeat
8      foreach c_i ∈ C do
           /* Calculate new position                */
           /* of all centroids                      */
9          updateCentroid(c_i);
10     end
11     foreach d_i ∈ D do
           /* Reassign entities to centroids        */
12         closest = closestCluster(d_i);
13         if closest ≠ m(d_i) then
14             m(d_i) = closest;
15         end
16     end
17 until convergenceReached();
```





if corresponding metric value is met

Track metrics per iteration

Create regression curve per metric

Apply convergence criterion on unseen data

Evaluation

Metric Performance

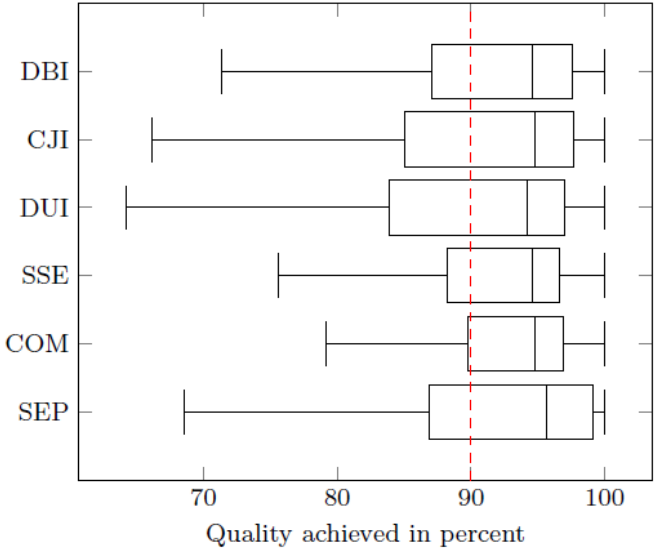


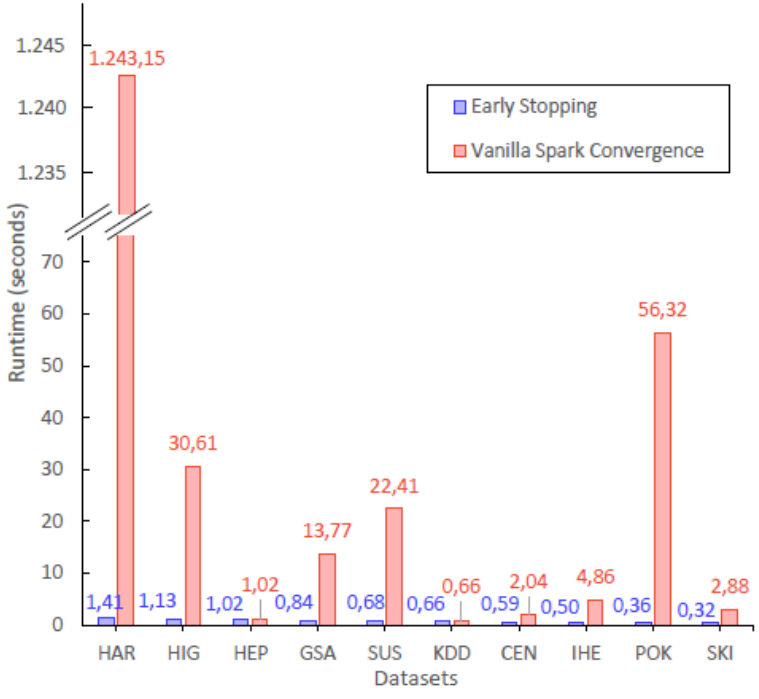
Fig. 2: Lower quartile, median and upper quartile per metric when demanding a quality of 90 %.

	Runtime (ms)			
Metric	MIN	MEDIAN	MAX	Averaged Overhead per Iteration
SEP	7.90e−05	2.99e−03	1.14	2.80e−08 %
COM	0.38	82.46	370.46	4.04e−04 %
SSE	0.38	84.16	825.51	4.06e−04 %
DUI	0.39	84.50	347.74	4.04e−04 %
CJI	0.38	84.81	562.14	4.05e−04 %
DBI	0.38	82.55	556.54	4.07e−04 %

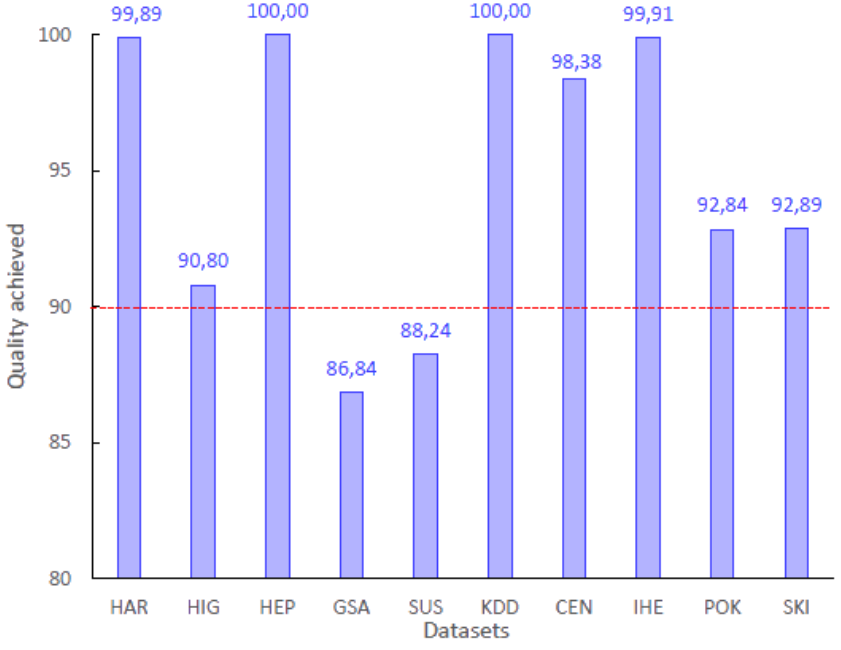
Table 4: Comparison of calculation time per metric throughout all iterations on the provided datasets.

Evaluation

Performance on Spark



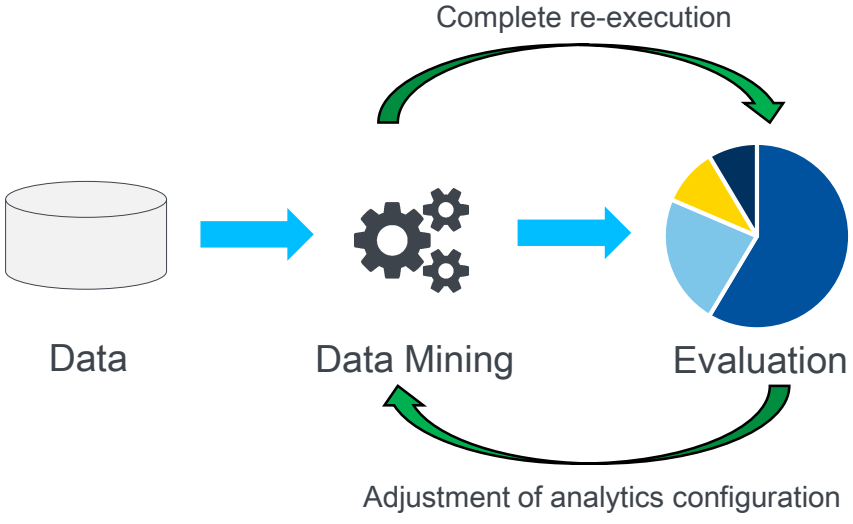
(a) Runtime comparison



(b) Quality reached when early stopping

Fig. 3: Comparison of vanilla Spark implementation of K-Means|| and early stopping when the change rate of SEP is below $-6.30e{-}02$, i.e., a quality of 90 % is demanded. Median values over 10 runs per dataset are depicted.

Conclusion & Outlook



- Considerable time savings (factor up to **800**!)
- Achieving qualitative demands regularly
- Spark Implementation available at: https://github.com/manuelfritz/EarlyStoppingKmeans

→ By how much can we accelerate the **whole** exploration process until a solid solution is achieved?

University of Stuttgart
Germany

Thank you!

Manuel Fritz

e-mail Manuel.Fritz@ipvs.uni-stuttgart.de
phone +49 (0) 711 685-88238
fax +49 (0) 711 685-78238

University of Stuttgart
Institute for Parallel and Distributed Systems
Universitätsstraße 38, 70569 Stuttgart, Germany