Data Stream Analysis

Francesco Marcelloni

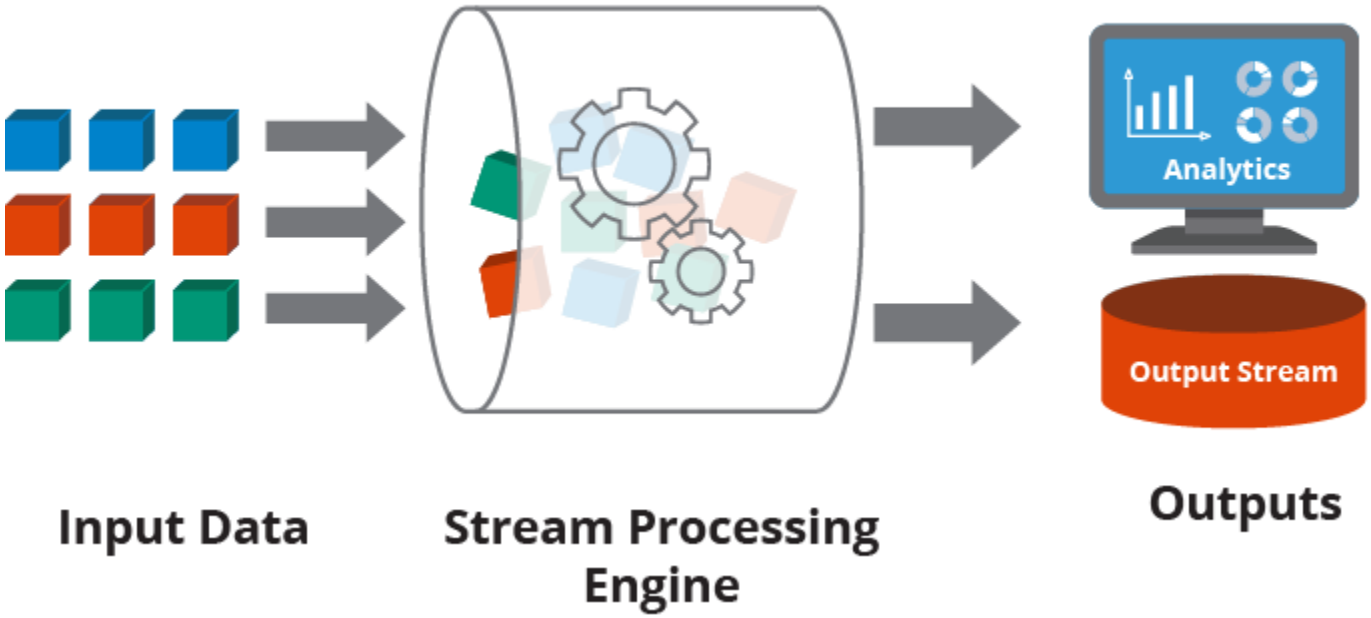
Department of Information Engineering
University of Pisa
ITALY

Chapter 13. Data Stream Analysis

- Introduction 
- Clustering
- Classification

What is a Data Stream?

- **Data stream**: a potentially **unbounded**, **ordered sequence of instances**. A data stream S may be shown as $S = \{x_1, x_2, x_3, ..., x_N\}$, where x_i is i-th data instance, which is a d-dimensional feature vector and *N* goes to infinity.



Problems in Mining Data Streams

- **Traditional Data Mining techniques usually require**
  - Entire dataset to be present
  - Multiple scans of the overall dataset
  - Random access to instances
  - Computationally heavy learning phases

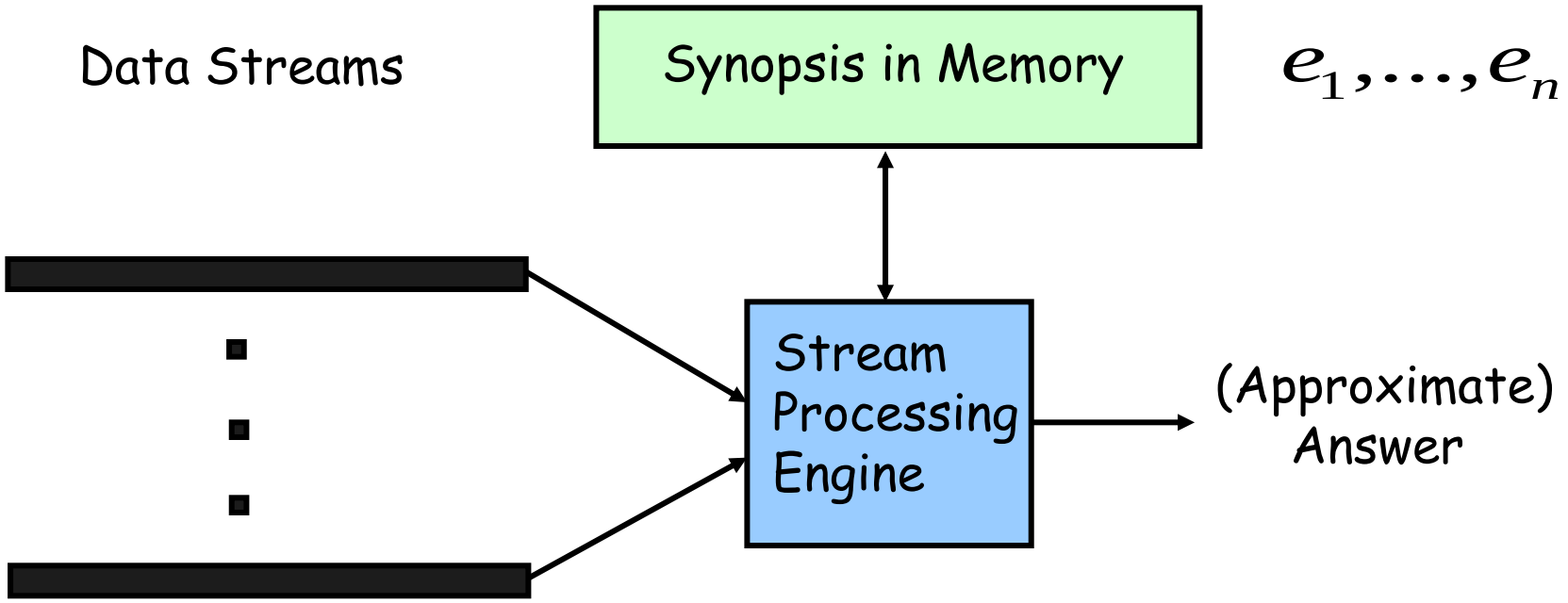
- **Challenges of stream mining**
  - Impractical (and impossible) to store the whole dataset
  - Impractical (and impossible) to perform multiple scans of the overall dataset
  - Random access is expensive
  - Simple calculation per data due to time and space constraints

Motivation

- **A growing number of applications generate streams of data**
  - Performance measurements in network monitoring and traffic management
  - Log records generated by Web Servers
  - Tweets on Twitter
  - Transactions in retail chains, ATM operations in banks
  - Sensor network data

- **Application characteristics**
  - Massive volumes of data
  - Records arrive at a rapid rate

Computational Model



- **Stream processing requirements**
  - **Single pass**: Each record is examined at most once
  - **Bounded storage**: Limited Memory (M) for storing synopsis
  - **Real-time**: Per record processing time (to maintain synopsis) must be low

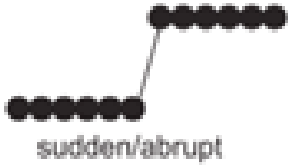
Data Stream Mining Algorithms

- **Generally, algorithms compute approximate answers**
  - Difficult to compute answers accurately with limited memory
- **Approximate answers - Deterministic bounds**
  - Algorithms only compute an approximate answer, but bounds on error
- **Approximate answers - Probabilistic bounds**
  - Algorithms compute an approximate answer with high probability
  - With probability at least 1-δ, the computed answer is within a factor ε of the actual answer
- **Single-pass algorithms for processing streams also applicable to (massive) terabyte databases!**

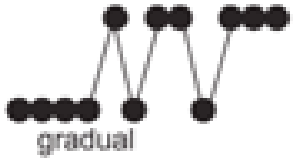
Concept Drift

- **Concept drift**: **unforeseen change in statistical properties of data stream instances over time**.
- There are four types of concept drift: sudden, gradual, incremental and recurring.

  - ***Sudden concept drift***: Between two consecutive instances, the change occurs at once, and after this time only instances of the new class are received.



  - ***Gradual concept drift***: The number of instances belonging to the previous class decreases gradually while the number of instances belonging to the new class increases over time. During a gradual concept drift, instances of both previous and new classes are visible.



Concept Drift

- ***Incremental concept drift***: Data instances belonging to the previous class evolves to a new class step by step. After the concept drift is completed, the previous class disappears. The instances that arrive during the concept drift are of transitional forms and they do not have to belong to either of the classes.

- ***Recurring concept drift***: The data instances change between two or more statistical characteristics several times. Neither of the classes disappears permanently but both of them arrive in turns.

Data Structures

- **It is not possible to store and manage the whole input data**
- Only a synopsis of the input stream is stored: special data structures enable to incrementally summarize the input stream.
- Four commonly used data structures

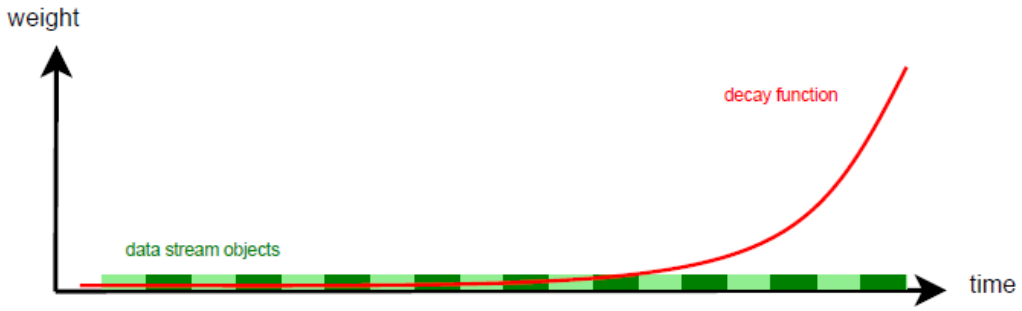
1. **Feature vectors**: summary of the data instances
2. **Prototype arrays**: keep only a number of representative instances that exemplify the data
3. **Coreset trees**: keep the summary in a tree structure
4. **Grids**: keep the data density in the feature space

The Window Model

- It is more efficient to process recent data instead of the whole data.
- **Damped window model**
  - Recent data have more weight than the older data: the importance of the instances decreases by time
  - Usually implemented using decay functions which scale down the weight of the instances, depending on the time passed since the instance is received

$$f(t) = 2^{-\lambda t}$$

where λ is the decay rate: Higher decay rate in the function means a more rapid decrease in the value

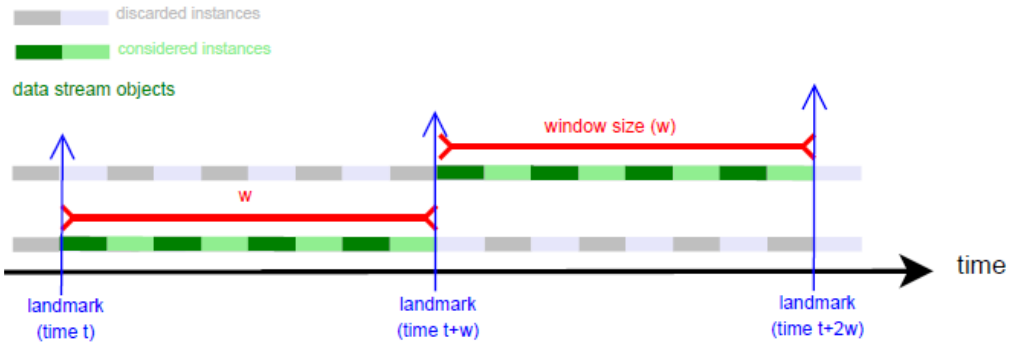


(a) Damped window model.

The Window Model

- **Landmark window model**
  - The whole data between two landmarks are included in the processing and all of the instances have equal weight
  - Consecutive windows do not intersect and the new window just begins from the point the previous window ends
  - Let *w* be the window length. Then, data instances belonging to the m-th window are calculated using

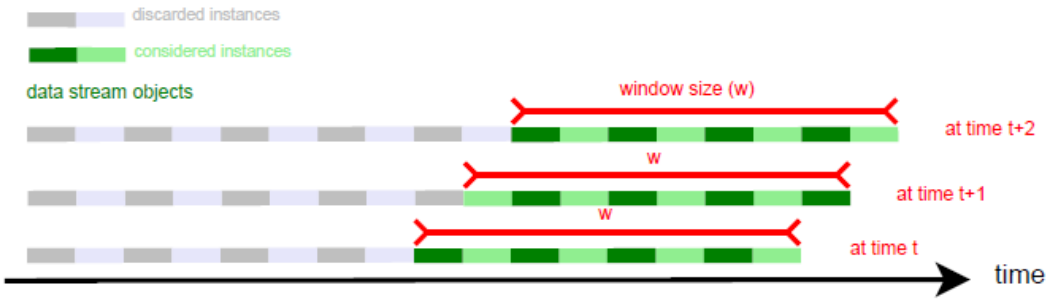
$$W_m = [x_{m*w}, \ldots, x_{(m+1)*w-1}] \qquad m = \left\lfloor \frac{i}{w} \right\rfloor$$



The Window Model

- **Sliding window model**
  - The window swaps one instance at each step: The older instance moves out of the window, and the most recent instance moves in to the window by FIFO style.
  - All instances in the window have equal weight and consecutive windows mostly overlap
  - Let *w* be the window length. Then, data instances belonging to the m-th window are calculated using

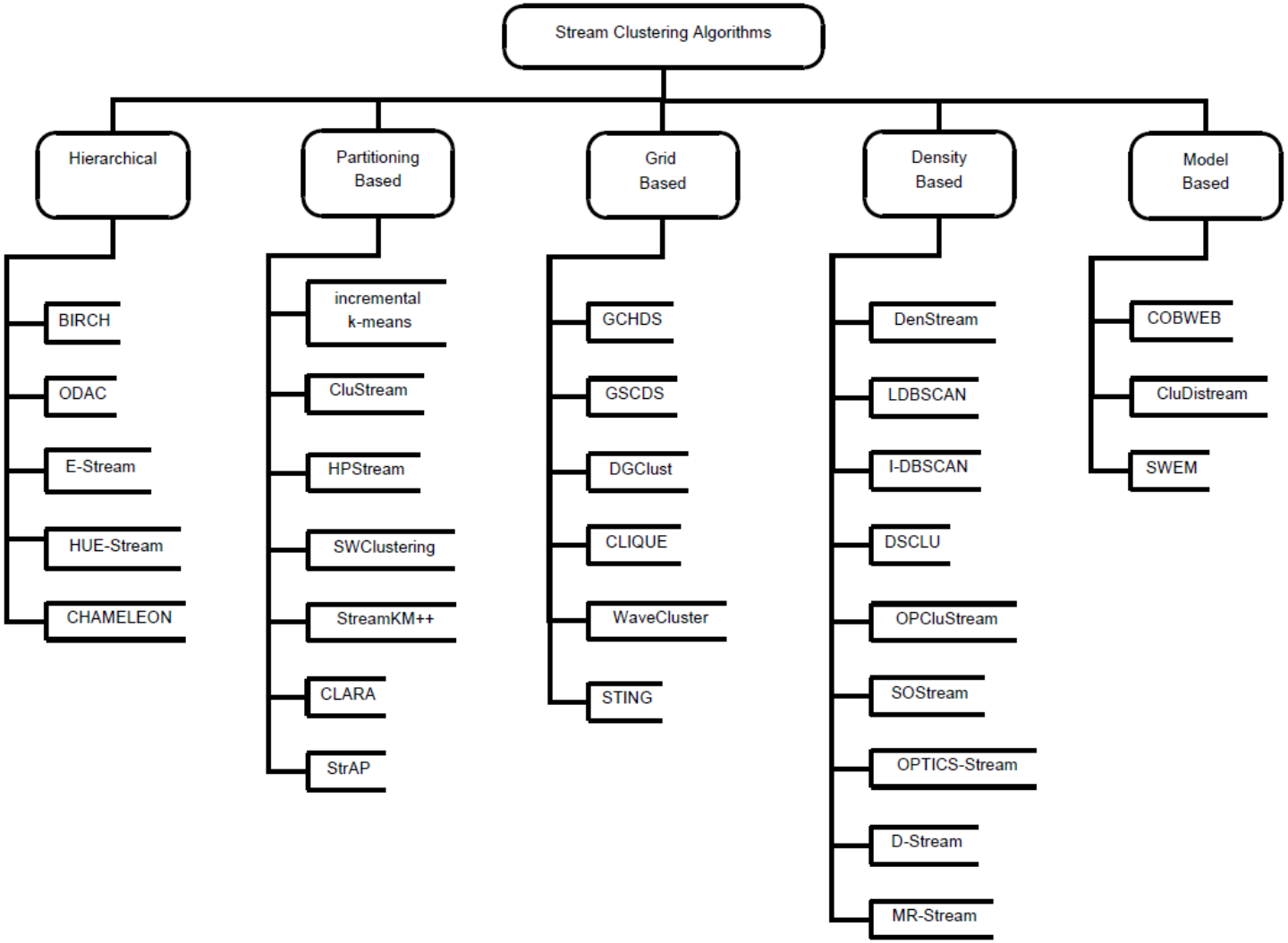
$$W_m = [x_m, ..., x_{(m+w-1)}]$$



Chapter 13. Data Stream Analysis

- Introduction
- Clustering
- Classification

Clustering


Stream Clustering Algorithms
Hierarchical
Partitioning Based
Grid Based
Density Based
Model Based
BIRCH
ODAC
E-Stream
HUE-Stream
CHAMELEON
incremental k-means
CluStream
HPStream
SWClustering
StreamKM++
CLARA
StrAP
GCHDS
GSCDS
DGClust
CLIQUE
WaveCluster
STING
DenStream
LDBSCAN
I-DBSCAN
DSCLU
OPCluStream
SOStream
OPTICS-Stream
D-Stream
MR-Stream
COBWEB
CluDistream
SWEM

Data Stream Clustering

- In most cases, **true class labels are not available** for stream instances and there is no prior knowledge about the number of classes.
- Therefore, clustering, being unsupervised is one of the most suitable data mining and data analysis methods for data streams.

- Zubaroğlu, A., Atalay, V. Data stream clustering: a review. Artif Intell Rev (2020). https://doi.org/10.1007/s10462-020-09874-x

Adaptive Streaming k-Means

- **Adaptive Streaming k-Means** (Puschmann D, Barnaghi P, Tafazolli R (2017) Adaptive clustering for dynamic iot data streams. IEEE Internet of Things Journal 4(1):64–74)

Number of data instances accumulated in initialization phase

Algorithm 1 streamingKMeans (S, l)

Input: S : the input data stream
Input: l : length of data sequence used for initialization
1: % Initialization phase
2: **for** *candidateCentroids* **in** *determineCentroids*(l number of data instances) **do**
3: run *kmeans* with *candidateCentroids*
4: calculate *silhouette coefficient* of the *kmeans* result
5: **end for**
6: keep *centroids* of the best clustering
7: % Continuous clustering phase
8: **loop**
9: **if** *changeDetected* on the input stream **then**
10: *re-initialize* the algorithm by running again the initialization phase
11: **end if**
12: run *kmeans* with last found, best centroids
13: **end loop**

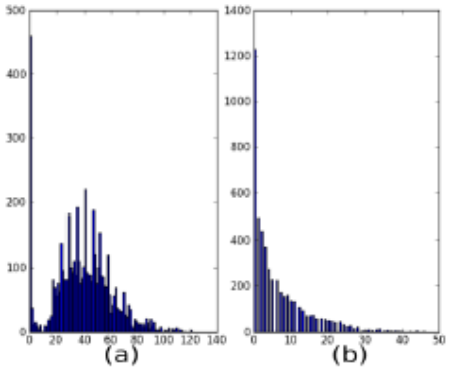
Adaptive Streaming k-Means

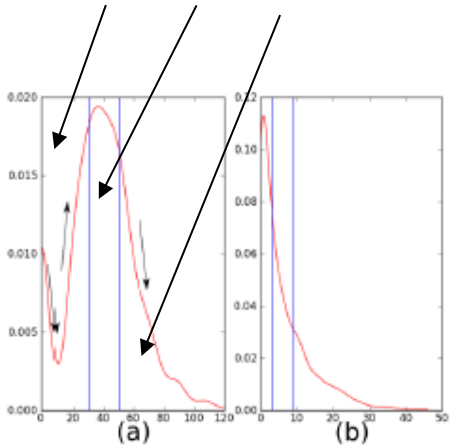
- **Initialization phase**
  - function *determineCentroids()* **finds *k* and determines candidate centroids**
    - Estimate the probability density function (PDF) of the data for each feature
    - Determine the directional changes of the PDF curves: each change identifies a new region. The region can be defined as the area between two consecutive directional changes of the PDF curve
    - **Number of regions is considered as a candidate *k* and centers of these regions are considered as candidate initial centroids**

Adaptive Streaming k-Means

- **Initialization phase**
  - function *determineCentroids()* in order to find *k* and determine candidate centroids
    - Different features generally show different distributions and different centroids
    - For each value of k, the PDF curve are split into equiprobable areas. The boundaries of these areas are called beta points. Since we are interested in the center of these region, the middle points between two adjacent betas are computed and saved as initial centroids.

areas of equiprobable distributions





Distribution

PDF

Adaptive Streaming k-Means

- **Initialization phase**
  - Different features generally show different distributions and different centroids

$$k \in [k_{min}, k_{min} + k_{max}]$$

  - The loop 2-5 is executed for these values of *k* and for candidate centroids
  - Clustering results of different *k* values are compared according to silhouette coefficient and best *k* is selected with its corresponding centroids.

Adaptive Streaming k-Means

- **Continuous clustering phase**
  - function *changeDetected()*
    - **standard deviation and mean of the input data are stored during the execution.**
    - The algorithm tracks how these two values change over time and predicts a concept drift according to the change.
    - When a **concept drift is predicted**, current cluster centroids are no longer valid. In such a case the concept drift is realized at line 9 and a re-initialization is triggered at line 10.

Adaptive Streaming k-Means

- **Complexity Analysis**
  - Let l be the length of the initial data sequence, and d be the data dimension.
  - Complexity of estimating k for a single dimension is O(l)
  - Since this estimation is performed for all dimensions, total k estimation complexity becomes O(d · l).
  - After determining initial centroids running k-means takes O(l · d · k · cs) since no iterations of the algorithm are needed, where cs is the number of different centroid sets.
  - Assigning a newly received data instance to the nearest cluster during the online phase is O(k).
  - As a result, total worst case complexity of the algorithm is O(k) + O(d · l) + O(l · d · k · cs), which equals to O(d · l) + O(l · d · k · cs).

MuDi-Stream

Amini A, Saboohi H, Herawan T, Wah TY (2016) Mudi-stream: A multi density clustering algorithm for evolving data stream. J Netw Comput Appl 59(C):370–385

- **MuDi-Stream is a hybrid algorithm based on both density based and grid based approaches.**
- Input data instances are clustered in a density-based approach and outliers are detected using grids.
- For Data synopsis, core mini-clusters are used.
- **Core mini-clusters are specialized feature vectors which keep weight, center, radius and the maximum distance from an instance to the mean**.
- In the online phase core mini-clusters are created and kept up to date for each new data instance.
- In the offline phase final clustering is executed over the core mini-clusters.

MuDi-Stream

Algorithm 3 MuDi-Stream online phase (S, α, λ, *gridGranularity*, G)

Input: S : the input data stream
Input: α : density threshold
Input: λ : decay rate
Input: *gridGranularity*
Input: G : total density grids for all dimensions

```
Initialize the grid structure
loop
    Read next data instance x from data stream S
    cmc_s = Find the nearest cmc to x
    if cmc_s involve x then
        Add x to cmc_s
    else
        Map x to the gird
        if Updated grid is dense enough then
            Create a cmc from updated grid
        end if
    end if
    if It is pruning period then
        Remove low weighted grids
        Remove low weighted cmcs
    end if
end loop
```

MuDi-Stream

Algorithm 4 MuDi-Stream offline phase (core mini-clusters)

Input: core mini-clusters

1: Mark all *cmcs* as unvisited
2: **repeat**
3: Randomly choose an unvisited *cmc*, called cmc_p
4: Mark cmc_p as visited
5: **if** cmc_p has *neighbors* **then**
6: Create new final cluster C
7: Add cmc_p to C
8: Add *neighbors* of cmc_p to C
9: **for each** *cmc* in C **do**
10: **if** *cmc* is unvisited **then**
11: Mark *cmc* as visited
12: Add *neighbors* of *cmc* to C
13: **end if**
14: **end for**
15: **else**
16: Mark cmc_p as *noise*
17: **end if**
18: **until** All *cmcs* are visited

MuDi-Stream

- Inside a loop, an unvisited core mini-cluster is randomly chosen at line 3 and marked as visited at line 4.
  - If this core mini-cluster has no neighbors, it is marked as noise at line 16.
  - If it has neighbors, a new final cluster is created with this core mini-cluster and its neighbors, at lines 6-8.
- After that, each unvisited core mini-cluster in the new created final cluster is marked as visited and its neighbors are added to the same final cluster, at lines 9-14.
- This loop continues until all core mini-clusters are marked as visited.

- MUDI-stream is not suitable for high dimensional data, which makes the processing time longer, because of the grid structure.
- Clustering quality of MuDi-Stream strongly depends on input parameters density threshold, decay rate for damped window model and grid granularity.
- These parameters require an expert knowledge about the data.

MuDi-Stream

- Complexity Analysis.
  - Complexity of this linear search on core miniclusters for each new data instance is O(c) where c is the number of core mini-clusters.
  - Let G be total density grids for all dimensions, which is exponential to the number of dimensions. Space complexity of the grid is O(log G) because the scattered grid are pruned during the execution. Moreover, time complexity of mapping a data instance to the grid is O(log log G) because the list of the grids is maintained as a tree.
  - During the pruning, all core mini-clusters and grids are examined. This makes time complexity of pruning O(c) for core mini-clusters and O(log G) for grids.
  - As a result, the overall time complexity of MuDi-Stream is O(c) + O(log log G) + O(c) + O(log G), which equals to O(c) + O(log G).

Comparison

Algorithm	Multi Density Clusters	High Dimensional Data	Outlier Detection	Drift Adaption	Expert Knowledge
Adaptive Streaming k-Means	Yes	Suitable	No	Yes	No
MuDi-Stream	Yes	Not suitable	Yes	Yes	Required
CEDAS	No	Suitable	Yes	Yes	Required
Improved Data Stream Clustering	No	Suitable	Yes	Yes	No
DBIECM	Yes (not multi size)	Suitable	No	Yes	Required
I-HASTREAM	Yes	Suitable	Yes	Yes	No

Chapter 13. Data Stream Analysis

- Introduction
- Clustering
- Classification 

Classification: decision trees

- VFDT algorithm
  - “Mining High-Speed Data Streams”, KDD 2000.
  - Pedro Domingos, Geoff Hulten

- CVFDT algorithm (window approach)
  - “Mining Time-Changing Data Streams”, KDD 2001.
  - Geoff Hulten, Laurie Spencer, Pedro Domingos

Classification: decision trees

- Classic decision tree learners assume all training data can be simultaneously stored in main memory
- Disk-based decision tree learners repeatedly read training data from disk sequentially
  - Prohibitively expensive when learning complex trees

- Goal: **design decision tree learners that read each example at most once, and use a small constant time to process it**

Classification: Key Observation

- In order to find the best attribute at a node, it may be sufficient to consider only a small subset of the training examples that pass through that node.
  - Given a stream of examples, **use the first ones to choose the root attribute**.
  - Once the root attribute is chosen, **the successive examples are passed down to the corresponding leaves, and used to choose the attribute there, and so on recursively**.

- **Use Hoeffding bound to decide how many examples are enough at each node**

- Warning: the approach analyzed in the subsequent phases has some theoretical problem, but you can find it implemented in several tools which manage data streams

Hoeffding Bound

- Let X a random variable varying in a range R
- Let us assume that we have n observations of X

- Let $\bar{x}$ be the average value of the n observations

- The Hoeffding bound states that with probability 1-δ, the mean $\bar{X}$ of X is at least $\bar{x} - \varepsilon$ where

$$\varepsilon = \sqrt{\frac{R^2 \ln 1/\delta}{2n}}$$

How do we use the Hoeffding Bound?

- Let $G(X_i)$ be the heuristic measure used to choose test attributes (e.g. Information Gain, Gini Index)
- X_A: the attribute with the highest attribute evaluation value after seeing *n* examples.
- X_B: the attribute with the second highest split evaluation function value after seeing n examples.

- **Given a desired δ: if $\Delta\overline{G} = \overline{G}(X_A) - \overline{G}(X_B) > \varepsilon$**, with $\varepsilon = \sqrt{\frac{R^2 \ln 1/\delta}{2n}}$ and $\mathrm{R} = \ln(\mathrm{c})$, where *c* is the number of classes, after seeing *n* examples at a node,
  - the Hoeffding bound guarantees the true $\Delta G \geq \Delta\bar{G} - \varepsilon > 0$, with probability 1- δ.
  - This node can be split using X_A and the succeeding examples will be passed to the new leaves.

Which problem?

- Problem:
  - Split measures, like information gain and Gini index, cannot be expressed as a sum S of elements Y_i.

- Actually, correct formula

$$\epsilon = C_{Gain}(K, N)\sqrt{\frac{\ln(1/\delta)}{2N}}$$

where

$$C_{Gain}(K, N) = 6(K \log_2 eN + \log_2 2N) + 2 \log_2 K$$

Rutkowski, L., Pietruczuk, L., Duda, P., Jaworski, M., Decision trees for mining data streams based on the McDiarmid's bound (2013) IEEE Transactions on Knowledge and Data Engineering, 25 (6), art. no. 6171195, pp. 1272-1279.

Two considerations

- Pre-pruning is carried out by considering at each node **a "null" attribute X_A** that consists of not splitting the node. Thus a split will be performed if, with confidence 1- δ, the best split found is better according to G than not splitting.

- The most significant part of the time cost per example is recomputing *G.*
  - **It is inefficient to recompute *G* for every new example**, because it is unlikely that the decision to split will be made at that specific point
  - Thus, VFDT (Very Fast Decision Tree) learner allows the user to specify a minimum number of new examples n_{min} that must be accumulated at a leaf before *G* is recomputed.

The Algorithm in Words

- **Calculate the information gain for the attributes and determines the best two attributes**
  - Pre-pruning: consider a "null" attribute that consists of not splitting the node
- At each node, check for the condition

$$\Delta\bar{G} = \bar{G}(X_A) - \bar{G}(X_B) > \varepsilon$$

- If condition is satisfied, create child nodes based on the test at the node
- If not, stream in more examples and perform calculations till condition is satisfied

The Algorithm in pseudo-code

Inputs:	S	is a sequence of examples,
	$\mathbf{X}$	is a set of discrete attributes,
	$G(.)$	is a split evaluation function,
	δ	is one minus the desired probability of choosing the correct attribute at any given node.
Output:	HT	is a decision tree.

The Algorithm in pseudo-code

Procedure HoeffdingTree $(S, \mathbf{X}, G, \delta)$
Let HT be a tree with a single leaf l_1 (the root).
Let $\mathbf{X_1} = \mathbf{X} \cup \{X_\emptyset\}$.
Let $\overline{G}_1(X_\emptyset)$ be the $\overline{G}$ obtained by predicting the most frequent class in S.
For each class y_k
  For each value x_{ij} of each attribute $X_i \in \mathbf{X}$
    Let $n_{ijk}(l_1) = 0$.
For each example $(\mathbf{x}, y_k)$ in S
  Sort $(\mathbf{x}, y)$ into a leaf l using HT.
  For each x_{ij} in $\mathbf{x}$ such that $X_i \in \mathbf{X}_l$
    Increment $n_{ijk}(l)$.
  Label l with the majority class among the examples seen so far at l.

n_{ijk} are the sufficient statistics needed to compute most heuristic measures

The Algorithm in pseudo-code

If the examples seen so far at l are not all of the same class, then
  Compute $\overline{G}_l(X_i)$ for each attribute $X_i \in \mathbf{X}_l - \{X_\emptyset\}$ using the counts $n_{ijk}(l)$.
  Let X_a be the attribute with highest $\overline{G}_l$.
  Let X_b be the attribute with second-highest $\overline{G}_l$.
  Compute ϵ using Equation 1.
  If $\overline{G}_l(X_a) - \overline{G}_l(X_b) > \epsilon$ and $X_a \neq X_\emptyset$, then
    Replace l by an internal node that splits on X_a.
    For each branch of the split
      Add a new leaf l_m, and let $\mathbf{X_m} = \mathbf{X} - \{X_a\}$.
      Let $\overline{G}_m(X_\emptyset)$ be the $\overline{G}$ obtained by predicting the most frequent class at l_m.
      For each class y_k and each value x_{ij} of each attribute $X_i \in \mathbf{X_m} - \{X_\emptyset\}$
        Let $n_{ijk}(l_m) = 0$.
Return HT.

Performance Analysis

- *p*: probability that an example passed through DT to level *i* will fall into a leaf at that point
- The **expected disagreement** between the tree produced by Hoeffding tree algorithm and that produced using infinite examples at each node **is no greater than δ / p.**
- Required memory: O(leaves * attributes * values * classes)

CVFDT

- **CVFDT (Concept-adapting Very Fast Decision Tree learner)**
  - Extend VFDT
  - Maintain VFDT's speed and accuracy
  - Detect and respond to changes in the example-generating process

- Observations
  - With a **time-changing concept**, the current splitting attribute of some nodes may not be the best any more.
  - An outdated subtree may still be better than the best single leaf, particularly if it is near the root.
    - Grow an alternative subtree with the new best attribute at its root, when the old attribute seems out-of-date.
  - Periodically use a bunch of samples to evaluate qualities of trees.
    - Replace the old subtree when the alternate one becomes more accurate.