Data Stream Mining

João Gama
jgama@fep.up.pt
Learning from Data Streams

Powerful Ideas
- Clustering Learning
- Predictive Learning
- Change
- Novelty Detection

Streaming Networks
- Evolving Networks
- Sampling
- Community Detection

Final Comments
Data Streams: Continuous flow of data generated at high-speed in Dynamic, Time-changing environments. We need to maintain Decision models in real time. Decision Models must be capable of:

- incorporating new information at the speed data arrives;
- detecting changes and adapting the decision models to the most recent information;
- forgetting outdated information;

Unbounded training sets, dynamic models.
1. One-pass algorithms: random access to data has high cost
2. Limited computational resources: time, memory, bandwidth
3. Anytime prediction
Powerful Ideas

- **Summarization:**
  Compact summaries to store sufficient statistics and fast update rules

- **Approximation:**
  How much data we need to learn an hypothesis $\hat{H}$ that with high probability, is within small error of the true hypothesis?

- **Monitoring the learning process: Change and Estimation**
Cluster Feature Vector

Birch: Balanced Iterative Reducing and Clustering using Hierarchies, by Zhang, Ramakrishnan, Livny 1996

Cluster Feature Vector: $\text{CF} = (N, LS, SS)$

- $N$: Number of data points
- $LS$: $\sum_1^N \bar{x}_i$
- $SS$: $\sum_1^N (\bar{x}_i)^2$

Constant space irrespective to the number of examples!
Micro clusters

The sufficient statistics of a cluster $A$ are $CF_A = (N, LS, SS)$.

- $N$, the number of data objects,
- $LS$, the linear sum of the data objects,
- $SS$, the sum of squared the data objects.

Properties:

- Centroid $= LS/N$
- Radius $= \sqrt{SS/N - (LS/N)^2}$
- Diameter $= \sqrt{2 \times N \times SS - 2 \times LS^2 / N \times (N-1)}$
Micro clusters

Given the sufficient statistics of a cluster $A$, $CF_A = (N_A, LS_A, SS_A)$. Updates are:

- **Incremental:** a point $x$ is added to the cluster:
  
  \[ LS_A \leftarrow LS_A + x; \quad SS_A \leftarrow SS_A + x^2; \quad N_A \leftarrow N_A + 1 \]

- **Additive:** merging clusters $A$ and $B$:
  
  \[ LS_C \leftarrow LS_A + LS_B; \quad SS_C \leftarrow SS_A + SS_B; \quad N_C \leftarrow N_A + N_B \]

- **Subtractive:**
  
  \[ CF(C_1 - C_2) = CF(C_1) - FV(C_2) \]
CluStream

*CluStream: A Framework for Clustering Evolving Data Streams*, Aggarwal, J. Han, J. Wang, P. Yu (VLDB03)

- Divide the clustering process into online and offline components
  - Online: periodically stores summary statistics about the stream data
    - Micro-clustering: better quality than k-means
    - Incremental, online processing and maintenance
  - Offline: answers various user queries based on the stored summary statistics
    - Tilted time framework: register dynamic changes
- With limited overhead to achieve high efficiency, scalability, quality of results and power of evolution/change detection
CluStream: Online Phase

Inputs:
- Maximum micro-cluster diameter $D_{\text{max}}$

For each $x$ in the stream:
- Find the nearest micro-cluster $M_i$
  - IF the diameter of $(M_i \cup x) < D_{\text{max}}$
  - THEN assign $x$ to that micro-cluster
    - $M_i \leftarrow M_i \cup x$
  - ELSE Start a new micro-cluster based on $x$
Pyramidal Time Frame

- The micro-clusters are stored at snapshots.
- The snapshots follow a pyramidal pattern
- The micro-clusters might be aggregated using tilted histograms

(a) Natural Tilted Time Window

(b) Logarithmic Tilted Time Window
Any Time Stream Clustering

*The CluStree: indexing micro-clusters for anytime stream mining*, Kranen, Assent, Baldauf, Seidl, KAIS 2011

Properties of anytime algorithms

- Deliver a model at any time
- Improve the model if more time is available
  - Model adaptation whenever an instance arrives
  - Model refinement whenever time permits
- an online component to learn micro-clusters
- Any variety of online components can be utilized
- Micro-clusters are subject to exponential aging
Analysis

- find the cluster structure in the current window,
- find the cluster structure over time ranges with granularity confined by the specification of window size and boundary,
- put different weights on different windows to mine various kinds of weighted cluster structures,
- mine the evolution of cluster structures based on the changes of their occurrences in a sequence of windows
Bibliography: Cluster data streams


- *CluStream: A Framework for Clustering Evolving Data Streams*, Aggarwal, J. Han, J. Wang, P. Yu VLDB03


- *An effective evaluation measure for clustering on evolving data streams*; Kremer, Kranen, Jansen, Seidl, Bifet, Holmes, Pfahringer, KDD 2011

Adaptive Learning Algorithms

A survey on concept drift adaptation, Gama, Zliobaite, Bifet et al, ACM-CSUR 2014
Decision Trees and Rules

One of the most used models for data mining.

- Nonparametric method
  Does not assume any particular distribution for the data. Can build models for any function given a sufficient number of training examples.

- The structure of the decision model is independent of the scale the variables. Monotone transformations of the variables ($\log x, 2 \times x, \ldots$) do not alter the structure of the model.

- High degree of interpretability A complex decision (predict the class value) is decomposed into a succession of elementary decisions.

- Robust to the presence of extreme points and attributes redundant or irrelevant. Selection mechanism attributes.
Learning Decision Trees

The base Idea

- Which attribute to choose at each splitting node?
- A small sample can often be enough to choose the optimal splitting attribute

- Collect sufficient statistics from a small set of examples
- Estimate the merit of each attribute

How large should be the sample?

- **The wrong idea:** Fixed sized, defined *apriori* without looking for the data;
- **The right idea:** Choose the sample size that allow to differentiate between the alternatives.
Very Fast Decision Trees

*Mining High-Speed Data Streams*, P. Domingos, G. Hulten; KDD 2000

The base Idea
A small sample can often be enough to choose the optimal splitting attribute

- Collect sufficient statistics from a small set of examples
- Estimate the merit of each attribute
- Use Hoeffding bound to guarantee that the best attribute is really the *best*.
  - Statistical evidence that it is better than the second best
Very Fast Decision Trees: Main Algorithm

- **Input:** $\delta$ desired probability level.
- **Output:** $T$ A decision Tree
- **Init:** $T \leftarrow$ Empty Leaf (Root)
- **While (TRUE)**
  - Read next example
  - Propagate example through the tree from the root till a leaf
  - Update sufficient statistics at leaf
  - If $\text{leaf}(\#\text{examples}) > N_{\text{min}}$
    - Evaluate the merit of each attribute
    - Let $A_1$ the best attribute and $A_2$ the second best
    - Let $\epsilon = \sqrt{R^2 \ln(1/\delta)/(2n)}$
    - If $G(A_1) - G(A_2) > \epsilon$
      - Install a splitting test based on $A_1$
      - Expand the tree with two descendant leaves
MAESTRA - Learning from Massive, Incompletely annotated, and Structured Data

VFDT

\[
\epsilon = \sqrt{\frac{R^2 \log(1/\delta)}{2N}}
\]

From Gehrke's SIGMOD tutorial slides
Hoeffding Algorithms

- **Classification:**
  - Mining high-speed data streams, P. Domingos, G. Hulten, KDD, 2000

- **Regression:**
  - *Learning model trees from evolving data streams*; Ikonomovska, Gama, Dzeroski; Data Min. Knowl. Discov. 2011

- **Rules:**
  - *Learning Decision Rules from Data Streams*, J. Gama, P. Kosina; IJCAI 2011

- **Clustering:**

- **Multiple Models:**
  - Ensembles of Restricted Hoeffding Trees. Bifet, Frank, Holmes, Pfahringer; ACM TIST; 2012

- ...
Rules

Problem: very large decision trees have context that is complex and hard to understand

- Rules: self-contained, modular, easier to interpret, no need to cover the universe

- $\mathcal{L}$ keeps sufficient statistics to:
  - make predictions
  - expand the rule
  - detect changes and anomalies
Adaptive Model Rules

Adaptive Model Rules from Data Streams, Almeida, Ferreira, Gama; ECML/PKDD 2013

- Ruleset: ensemble of rules + default rule
- Rule prediction: mean, linear model
- Ruleset prediction:
  - Ordered: only first rule covers instance
  - Unordered: weighted avg. of predictions of rules covering instance $\mathbf{x}$
  - Weights inversely proportional to error

$$\hat{f}(\mathbf{x}) = \sum_{R_i \in S(\mathbf{x}_i)} \theta_i \hat{y}_i,$$

E.g: $\mathbf{x} = [4, -1, 1, 2]$
AMRules Induction

- Rule creation: default rule expansion
- Rule expansion: split on attribute maximizing $\sigma$ reduction
  - Hoeffding bound
    \[ \epsilon = \sqrt{R^2 \ln(1/\delta)/(2n)} \]
  - Expand when
    \[ \sigma_{1st}/\sigma_{2nd} < 1 - \epsilon \]
- Evict rule when P-H signals an alarm
- Detect and explain local anomalies

```
Algorithm 1: Training AMRules
Input: S: Stream of examples
begin
    \( R \leftarrow \{\}, D \leftarrow 0 \)
    foreach \((x, y) \in S\) do
        foreach Rule \( r \in S(x)\) do
            if \(-\text{IsAnomaly}(x, r)\) then
                if \(\text{PHTest}(\text{error}_r, \lambda)\) then
                    Remove the rule from \( R \)
                else
                    Update sufficient statistics \( L_r \)
                    ExpandRule(r)
            else
                ExpandRule(D)
                if \( D \) expanded then
                    \( R \leftarrow R \cup D \)
                    \( D \leftarrow 0 \)
        return \((R, L_D)\)
```
MT AMRules: subsets of related outputs

<table>
<thead>
<tr>
<th>Rule 1</th>
<th>Rule 2</th>
<th>...</th>
<th>Rule r</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_4 &gt; 1$</td>
<td>$X_2 &gt; 1$</td>
<td>$X_3 &gt; 0$</td>
<td>$X_3 \leq 5$</td>
<td></td>
</tr>
<tr>
<td>$X_2 \leq 0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X_1 &gt; 2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$\mathcal{L}_1 \downarrow \{y_1, y_3\}$  $\mathcal{L}_2 \downarrow \{y_2, y_3\}$  $\mathcal{L}_r \downarrow \{y_1\}$  $\mathcal{L}_D \downarrow \{y_1, y_2, y_3\}$

$\{y_1, y_2, y_3\}$

- Each rule makes predictions for a **subset of the output attributes**.
- The rule set predicts for **all output attributes**.
- Take advantage of the rules’ modularity model dependencies between output attributes.
Hoeffding Algorithms: Analysis

The number of examples required to expand a node only depends on the Hoeffding bound: $\epsilon$ decreases with $\sqrt{N}$.

- Low variance models:
  Stable decisions with statistical support.

- Low overfitting:
  Examples are processed only once.

- No need for pruning;
  Decisions with statistical support;

- **Convergence**: Hoeffding Algorithms becomes asymptotically close to that of a batch learner. The expected disagreement is $\delta/p$; where $p$ is the probability that an example fall into a leaf.
Bibliography on Predictive Learning

- *Mining High Speed Data Streams*, by Domingos, Hulten, SIGKDD 2000.
- *Learning decision trees from dynamic data streams*, Gama, Medas, and Rodrigues; SAC 2005
- *Learning model trees from evolving data streams*, Ikonomovska, Gama, Dzeroski; Data Min. Knowl. Discov. 2011
Change Detection in Predictive Learning

When there is a change in the class-distribution of the examples:

- The current model does not correspond any more to the current distribution.
- The error-rate increases

Base idea
Learning from data streams is a continuous process. Monitor the quality of the learning process using quality control techniques.

Main Problems:

- How to distinguish Change from Noise?
- How to React to drift?
Dynamic Window

Definition

- Novelty Detection refers to the automatic identification of unforeseen phenomena embedded in a large amount of normal data.

- **Novelty** is a relative concept with regard to our current knowledge:
  - It must be defined in the context of a representation of our current knowledge.

- Specially useful when novel concepts represent abnormal or unexpected conditions
  - Expensive to obtain abnormal examples
  - Probably impossible to simulate all possible abnormal conditions
One-Class Classification
Autoassociator Networks

*Concept-learning in the absence of counter-examples: an autoassociation-based approach* Nathalie Japcowitz, 1999

- Three layer network
- The nr. of neurons in the output layer is equal to the input layer
- Train the network such that $\hat{y}$ is equal to the $\hat{x}$
- The network is trained to reproduce the input at the output layer
Autoassociator Networks

To classify a test example $\vec{x}$

- Propagate $\vec{x}$ through the network and let $\vec{y}$ be the corresponding output;
- If $\sum_{i}^{k} (x_i - y_i)^2 < \textit{Threshold}$ Then the example is considered from class \textit{normal};
- Otherwise, $\vec{x}$ is a counter-example of the \textit{normal} class.
Novelty detection

- Training set (Offline Phase)
  - $D_{tr} = (X_1, y_1), (X_2, y_2), \ldots, (X_m, y_m)$
  - $X_i$: vector of input attributes for the $i$th example
  - $y_i$: target attribute
  - $y_i \in Y_{tr}$ where $Y_{tr} = c_1, c_2, \ldots, c_L$

- When new data arrive (Online Phase)
  - Given a sequence of unlabelled examples $X_{new}$
  - Goal: Classify $X_{new}$ in $Y_{all}$ where $Y_{all} = c_1, c_2, \ldots, c_L, \ldots, c_K$ and $K > L$

Open-set recognition
Novelty Detection Systems

- ECSMiner: Assume that the class label of new examples is known
- OLINDDA: unsupervised, but restricted to binary classification problems
- MINAS (MultI-class learNing Algorithm for data Streams)
  - Does not use the class labels of new examples
  - Can deal with novelty detection in data streams multi-class problem
Minas algorithm

MINAS: Multiclass Learning Algorithm for Novelty Detection in Data Streams, E. Faria, J. Gama, A. Carvalho, DAMI, 2016

- Unsupervised algorithm for novelty detection in data streams multi-class problems
  Represents each known class by a set of hyperspheres
- Use of offline (training) and online phases
  In each phase learns one or more classes
- Cohesive set of examples is necessary to learn new concepts or extensions
  Isolated examples are not considered as novelty
Evaluation

<table>
<thead>
<tr>
<th></th>
<th>C₁</th>
<th>C₂</th>
<th>N₁</th>
<th>N₂</th>
<th>N₃</th>
<th>N₄</th>
<th>Desconhecido</th>
</tr>
</thead>
<tbody>
<tr>
<td>C₁</td>
<td>6000</td>
<td>2900</td>
<td>0</td>
<td>800</td>
<td>50</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>C₂</td>
<td>2000</td>
<td>5200</td>
<td>700</td>
<td>0</td>
<td>30</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>C₃</td>
<td>300</td>
<td>450</td>
<td>1950</td>
<td>100</td>
<td>100</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>C₄</td>
<td>200</td>
<td>300</td>
<td>1900</td>
<td>1400</td>
<td>100</td>
<td>20</td>
<td>100</td>
</tr>
</tbody>
</table>

Matriz de Confusão

Grafo bipartido correspondente

Subgrafo bipartido resultante
Novelty Detection Bibliography

- Masud, Gao, Khan, Han, and Thuraisingham, *Classification and novel class detection in concept-drifting data streams under time constraints*, TKDE 2011
- Spinosa, Carvalho, Gama: *OLINDDA: a cluster-based approach for detecting novelty and concept drift in data streams* SAC 2007
- MINAS: Multiclass Learning Algorithm for Novelty Detection in Data Streams, E. Faria, J. Gama, A. Carvalho, DAMI, 2016
- D. Cardoso and F. França *A Bounded Neural Network for Open Set Recognition*, IJCNN 2015
Outline

Learning from Data Streams
- Powerful Ideas
- Clustering Learning
- Predictive Learning
- Change
- Novelty Detection

Streaming Networks
- Evolving Networks
- Sampling
- Community Detection

Final Comments
Social Networks as Dynamic Structures

- **Social networks** are a hot topic and a focus of considerable attention in recent research.
- Growing **availability of large volumes of relational data**, boosted by the proliferation of social media web sites.
- Number of social entities and the interactions among social entities **change over time** → **social networks have a dynamic nature**.
- One way to uncover the evolution patterns of social networks is **by monitoring the evolution of their communities**.
Event-based Dynamic Community Mining

- Dynamic communities are **unstable patterns** that can evolve in both membership and content.
- Dynamic communities undergo a **succession of events** during their life-cycle in the network.

**Community Evolution Events**

According to Palla et al. (2007)
Event-based Dynamic Community Mining

How can we perform the mapping of communities between different snapshots of the dynamic network?

- **Proposed solution:** compute conditional probabilities for each pair of communities found at consecutive time points

\[
P(X \in \text{Com}^u_{t_{i+\Delta t}} | X \in \text{Com}^m_{t_i}) = \\
= \frac{\sum P(x \in \text{Com}^m_{t_i} \cap \text{Com}^u_{t_{i+\Delta t}})}{\sum P(x \in \text{Com}^m_{t_i})}
\]
Event-based Dynamic Community Mining
Telco Data

Using Louvain Algorithm for Community Detection
Graph streams

-Anonymous CDR’s from Telecommunication networks Data Stream over 31 days
-Approx. 8 millions to 16 millions calls per day
-Night: Tens of calls/second; Day: hundreds of calls per second

Call-Graph Semantics

- Nodes as callers and callee
- Edges as calls
- Multi-graph with repetitive edges (calls between same nodes)
- Multi-graph mapped to weighted network as frequency of edges Incoming and outgoing calls as bi-directional edges
Sampling methods and Algorithms

- Which sampling methods preserve community structure?
- Which sampling methods maintain the properties of dynamic stream?
- Which are the most efficient algorithms?
- Which samples are biased towards some of the metrics?
- Which samples exhibit similar degree distribution as snapshot of stream?

Sampling Algorithms:
- Reservoir Sampling
- Space Saving
- Biased Random Sampling
Reservoir Sampling

Randomly choosing a sample of $k$ items from a list $S$ containing $n$ items.

Replaces elements with gradually decreasing probability
All the elements are chosen with same probability

- Fill the reservoir of size $k$ with first $n$ elements
- For each element $i$ after $n$
  - Generate a random number $r$ between 0 and $i - 1$
  - If $r < k$; Let $j$ be the element at position $r$ in $k$
  - Replace element $j$ with $i$
  - Else
  - Skip $I$

(Vitter 1985)
Space Saving Algorithm

Approximate approach for finding most frequent items

Maintain partial information of interest; monitor only a subset m of elements

- For each element e in stream
  - If e is monitored: Increment Count
  - Else
    - Let m be the element with least hits min
    - Replace m with e with count = min+1

(Metwally et al. 2005)
Biased Random Sampling

Generates a random sample by inserting every element from the stream with equal probability

Replacing elements from sample randomly

Biased towards recent elements in stream

- Fill the list of size $k$ with first $n$ elements
- For each element $i$ after $n$
  - Generate a random number $r$ between $0$ and $k$
  - Let $j$ be the element at position $r$ in $k$
  - Replace $j$ with $i$

*(Aggarwal 2006)*
SSE

Sample of size $10^4$ edges over 31 days using Fruchterman Reingold Layout
RS displays Least community structure
Sample of nodes with different modularities
Conclusions

- Sampling edges versus sampling nodes.
- Biased Reservoir Sample and Space Saving: stronger communities
- BRS and SSE samples display high degree centralities
- Reservoir sample and Biased Reservoir Sample show better performance with runtime
Community Detection

- Groups of nodes highly connected between them (Girvan and Newman, 2002)
  - Social networks include community groups (the origin of the term, in fact) based on common location, interests, occupation, etc.[5]
  - Metabolic networks have communities based on functional groupings.
  - Citation networks form communities by research topic

- Algorithms
  - Minimum-cut method (Ahuja et al., 1993)
  - Maximize modularity (Blondel et al. 2008)
The Louvain Algorithm

- Static Community Detection:
  - also called the Louvain Method ([BGLL08])
  - based on heuristics to optimise the modularity function
  - able to handle and discover communities in larger static networks
  - written in C++ source code freely available ¹

- Examples of large networks:
  - Twitter social network dataset with 2.4M nodes and 38M links
  - LinkedIn social network dataset with 21M nodes
  - Mobile phone network with 4M nodes and 100M links

- This talk:
  Dynamic community detection in evolving networks using locality modularity optimization, Social Network Analysis and Mining, 2016 [CSG16]

¹https://sites.google.com/site/findcommunities/
Modularity function

- Modularity is a scale value that measures the density of edges inside communities to edges outside communities.

\[
Q = \frac{1}{2m} \sum_{i,j} \left[ A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j) \tag{1}
\]

where \( A_{ij} \) represents the weight of the edge between \( i \) and \( j \), \( k_i = \sum_j A_{ij} \) is the sum of the weights of the edges attached to vertex \( i \), \( c_i \) is the community to which vertex \( i \) is assigned, the \( \delta \)-function \( \delta(u, v) \) is 1 if \( u = v \) and 0 otherwise and \( m = \frac{1}{2} \sum_{ij} A_{ij} \).

- Optimizing Modularity results in the best possible grouping of the nodes of a given network.

- However going through all possible iterations of the nodes into groups is impractical so heuristic algorithms are used.

- Louvain Method is a greedy method. It starts with small communities, then each small community is grouped into one node and the first step is repeated.
Algorithm steps

(a) original network

(b) initial communities

(c) step 1 of 1\textsuperscript{st} iteration

(d) step 2 of 1\textsuperscript{st} iteration

(e) step 1 of 2\textsuperscript{nd} iteration

(f) step 2 of 2\textsuperscript{nd} iteration

Figure: The original Louvain algorithm steps.
Limitations when adding new edges

Any network change implies a full rerun:

- Don’t allow tracking of communities:
  - non-deterministic
  - different solutions with multiple runs

**Figure:** Steps for Fig.1a with edges \{(1, 4), (5, 7), (1, 9), (10, 11)\} added.

(a) original network

(b) initial communities

(c) step 1 of 1\textsuperscript{st} iteration

(d) step 2 of 1\textsuperscript{st} iteration
Types of edges that may change the network

(a) Cross community edge

(b) Inner community edge

(c) Half-new edge / Edge to isolated node

(d) New edge / Edge between isolated nodes

Figure: Types of edges that can be added to or removed from a graph.
Cross Community Edge

- Adding cross community edge:
  - Two candidate operations can be taken:
    - Keeping the community structure unchanged
    - Combining both communities into one
  - Removing cross community edge:
    - Never result in the merging of existent communities neither result in the disband of any of the communities were the nodes linked by the removed edge belong. In this case the community structure will keep unchanged;
Adding inner community edge:
- Increases the inner connections of the community and keeps the inter-community connections unchanged.

Removing inner community edge:
- Decreases the inner connections of the community but keeps the inter-community connections unchanged. May lead to:
  - Keep the community structure unchanged
  - Dividing the community into smaller communities
Adding half-new edge:

- Two candidate operations can be taken: operation 3 or operation 4. The former assigns the new node the existent community, while the latter creates a new community with the new node;

Removing edge to isolated node:

- The removed node is a terminal node and therefore it will not change the inner connections of the community, so operation 7 should be taken;
Adding new edge:

- Two candidate operations can be taken: operation 3 or operation 4. The former assigns both nodes to the same community, while the latter creates two distinct communities for each one of the nodes;

Removing edge between isolated nodes:

- Removing isolated nodes only can extinguish the community those nodes belong to by applying operation 8. The other communities will remain unaffected;
Low Level / Upper Level Network

The algorithm uses two separated networks:

- The lower level network
  - where all the original nodes and final communities are maintained;
  - should be updated every time node community assignments change;
  - represented in the left stack of graphs

- The upper level network
  - used to perform the iterations of the Louvain algorithm;
  - requires to be updated when the addition and removal of nodes is performed in the lower level network;
  - represented in the right stack of graphs

- Both
  - the aggregated communities are represented by the node with higher order number (i.e.: a community of nodes 1, 3, 4, 7 and 9 is represented by community 9);
  - should be kept synced regarding the assignment of nodes to communities (community structure) and regarding network structure (nodes and edges).
Low Level / Upper Level Network

(e) add edge

(f) disband communities

(g) update from Louvain

(h) final communities

(i) add edge

(j) update communities

(k) Louvain Step 1

(l) Louvain Step 2
Outline

Learning from Data Streams
  Powerful Ideas
  Clustering Learning
  Predictive Learning
  Change
  Novelty Detection

Streaming Networks
  Evolving Networks
  Sampling
  Community Detection

Final Comments
Lessons Learned

Learning from data streams:

- Learning is not *one-shot*: is an evolving process;
- We need to monitor the learning process;
- Opens the possibility to reasoning about the learning
Reasoning about the Learning Process

Intelligent systems must:

- be able to adapt continuously to changing environmental conditions and evolving user habits and needs.
- be capable of predictive self-diagnosis.

The development of such self-configuring, self-optimizing, and self-repairing systems is a major scientific and engineering challenge.