Mining Data that Changes

17 July 2014
Data is Not Static

• Data is not static
  • New transactions, new friends, stop following somebody in Twitter, ...
  • But most data mining algorithms assume static data
    • Even a minor change requires a full-blown re-computation
Types of Changing Data

1. New observations are added
   - New items are bought, new movies are rated
   - The existing data doesn’t change

2. Only part of the data is seen at once

3. Old observations are altered
   - Changes in friendship relations
Types of Changing-Data Algorithms

• **On-line** algorithms get new data during their execution
  • Good answer at any given point
  • Usually old data is not altered

• **Streaming** algorithms can only see a part of the data at once
  • Single-pass (or limited number of passes), limited memory

• **Dynamic** algorithms’ data is changed constantly
  • More, less, or altered
Measures of Goodness

- **Competitive ratio** is the ratio of the (non-static) answer to the *optimal* off-line answer
  - Problem can be NP-hard in off-line
  - What’s the cost of uncertainty
- **Insertion** and **deletion times** measure the time it takes to update a solution
- **Space complexity** tells how much space the algorithm needs
Concept Drift

• Over time, users’ opinions and preferences change
  • This is called **concept drift**

• Mining algorithms need to counter it
  • Typically data observed earlier weights less when computing the fit
On-Line vs. Streaming

On-line

• Must give good answers at all times
• Can go back to already-seen data
• Assumes all data fits to memory

Streaming

• Can wait until the end of the stream
• Cannot go back to already-seen data
• Assumes data is too big to fit to memory
On-Line vs. Dynamic

On-line

- Already-seen data doesn’t change
- More focused on competitive ratio
- Cannot change already-made decisions

Dynamic

- Data is changed all the time
- More focused on efficient addition and deletion
- Can revert already-made decisions
Example: Matrix Factorization

• On-line matrix factorization: new rows/columns are added and the factorization needs to be updated accordingly.

• Streaming matrix factorization: factors need to be built by seeing only a small fraction of the matrix at a time.

• Dynamic matrix factorization: matrix’s values are changed (or added/removed) and the factorization needs to be updated accordingly.
On-Line Examples

• Operating systems’ cache algorithms
• Ski rental problem
• Updating matrix factorizations with new rows
  • I.e. LSI/pLSI with new documents
Streaming Examples

• How many distinct elements we’ve seen?
• What are the most frequent items we’ve seen?
• Keep up the cluster centroids over a stream
Dynamic Examples

• After insertion and deletion of edges of a graph, maintain its parameters:
  • Connectivity, diameter, max. degree, shortest paths, ...

• Maintain clustering with insertions and deletion
Streaming
Sliding Windows

- Streaming algorithms work either per element or with **sliding windows**
  - Window = last $k$ items seen
    - Window size = memory consumption
    - “What is $X$ in the current window?”
Example Algorithm: The 0th Moment

- **Problem**: How many distinct elements are in the stream?
  - Too many that we could store them all, must estimate
  - Idea: store a value that lets us estimate the number of distinct elements
  - Store many of the values for improved estimate
The Flajolet–Martin Algorithm

• Hash element $a$ with hash function $h$ and let $R$ be the number of trailing zeros in $h(a)$

• Assume $h$ has large-enough range (e.g. 64 bits)

• The estimate for # of distinct elements is $2^R$

• Clearly space-efficient

• Need to store only one integer, $R$

Does Flajolet–Martin Work?

• Assume the stream elements come u.a.r.

• Let $\text{trail}(h(a))$ be the number of trailing 0s

• $\Pr[\text{trail}(h(a)) \geq r] = 2^{-r}$

• If stream has $m$ distinct elements, $\Pr[\text{“For all distinct elements, } \text{trail}(h(a)) \leq r\text{”}] = (1 - 2^{-r})^m$
  
  • Approximately $\exp(-m2^{-r})$ for large-enough $r$

  • Hence: $\Pr[\text{“We have seen a s.t. } \text{trail}(h(a)) \geq r\text{”}]$
  
  • approaches 1 if $m \gg 2^r$ and approaches 0 if $m \ll 2^r$
Many Hash Functions

• Take average?
  • A single $r$ that’s too high at least doubles the estimate
    ⇒ the expected value is infinite

• Take median?
  • Doesn’t suffer from outliers
  • But it’s always a power of two
    ⇒ adding hash functions won’t get us closer than that

• Solution: group hash functions in small groups, take their average and the median of the averages
  • Group size preferably $\approx \log m$
Example Dynamic Algorithm
Users and Tweets

• Users follow tweets
• A bipartite graph
• We want to know (approximate) bicliques of users who follow similar tweeters
### Boolean Matrix

The boolean matrix is a 6x6 matrix that represents the logical relationships between the elements A, B, C, D, and E. Each element in the matrix is either 0 or 1, indicating the presence or absence of a connection.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The diagram on the right visualizes these connections, with nodes representing the elements and edges indicating the logical relationships as defined by the matrix.
Boolean Matrix Factorizations

\begin{align*}
\begin{bmatrix}
1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}
\approx
\begin{bmatrix}
1 & 0 \\
1 & 0 \\
0 & 0 \\
0 & 1 \\
0 & 1 \\
0 & 0
\end{bmatrix}
\approx
\begin{bmatrix}
1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0
\end{bmatrix}
\end{align*}
Boolean Matrix Factorizations

\[
\begin{array}{cccccc}
1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
\end{array}
\]

\[
\begin{array}{cccccc}
1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccc}
1 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
Fully Dynamic Setup

• Can handle both addition and deletion of vertices and edges
  • Deletion is harder to handle
• Can adjust the number of bicliques
  • Based on the MDL principle
This Ain’t Prediction

- The goal is not to predict new edges, but to adapt to the changes
- The quality is computed on observed edges
- Being good at predicting helps adapting, though
First Attempt

- Re-compute the factorization after every addition
- Too slow
- Too much effort given the minimal change
Example

\[
\begin{array}{cccc}
1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 \\
\end{array}
\approx
\begin{array}{cccc}
1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 \\
\end{array}
\]
### Step 1: Remove

\[
\begin{array}{cccc}
1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0
\end{array}
\]

\[
\begin{array}{cccc}
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{array}
\]

\[
\begin{array}{cccc}
1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0
\end{array}
\]
Step 2: Add

\[
\begin{array}{cccccc}
1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 \\
\end{array}
\approx
\begin{array}{cccccc}
1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
### Step 3: Remove

![Image of a matrix with marked entries being removed]

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>≈</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

![Image of a matrix with a single entry removed]

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Step 4: Add
Step 5: Add

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

≈

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Step 6: Remove

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

≈

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
One Factor Too Many?
Adjusting the rank

• Use the MDL principle: Best rank is the one that lets us encode the data with least number of bits
  • Encode the data matrix using the factors and the residual (error) matrix
  • Remove a factor if doing so reduces the overall encoding length
• Adding a factor is harder: need to have a new candidate factor to add
Adding a new factor

• Checking if we should remove a factor is easy
• But how to decide should we add a factor?
  • We need to decide what kind of a factor to add
  • Simple heuristic: build candidates based on not-yet covered 1s and select the one with largest area
Making global updates

- The basic algorithm makes only somewhat local updates
- For global updates, we iteratively update $B$ and $C$
  - Fix $B$, update $C$; fix $C$, update $B$; etc.
- The problem is (still) NP-hard – we use a heuristic
- Computationally expensive
Error Over Time

![Graph showing error over time with a linear trend line and a green line with a zigzag pattern.](image)
Empirical Competitiveness

![Bar chart showing competitiveness of Delicious, LastFM, and Movielens datasets with 'dynamic' and 'w/ iterations' conditions.]
# Running Times

<table>
<thead>
<tr>
<th></th>
<th>Delicious</th>
<th>LastFM</th>
<th>Movielens</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offline</td>
<td>43</td>
<td>200</td>
<td>4,21</td>
</tr>
<tr>
<td>Dynamic</td>
<td>4</td>
<td>213</td>
<td>4,452</td>
</tr>
<tr>
<td>w/ iterations</td>
<td>585</td>
<td>1,504</td>
<td>11,295</td>
</tr>
</tbody>
</table>
The MDL-optimal rank of the data, for example, is over 300.

Such low ranks are not universal, though. Over time, the dynamic algorithm reduces the rank to one. That the offline method is better than the comparable online method – the only reasonable alternative – is rather surprising, but as we have seen, with the heuristics involved here, it sometimes is the case.

The two algorithms do not constantly agree with each other. The offline algorithm agrees with the dynamic algorithm more often. That the offline algorithm agrees with the dynamic algorithm more often than it behaves very well when the it is allowed to adjust the rank.

The ranks alone do not tell the whole story, however. Per example, have been proposed. The characteristics and behaviour of such methods are very different.

From Figure 4 we see that while the offline algorithm changes its rank more often.

Figure 4: Description lengths of the dynamic and offline algorithm.

When the number of tiles is given and the goal is to minimize the number of uncovered items, the problem is more akin to boolean matrix factorization dealing with binary matrices. Methods to obtain better boolean rank and description lengths for the algorithms are presented in Figure 4.

The data then has extremely low rank. Why is that? Looking at the data computed after every 100 changes.

In experiments done with the other data sets (results not shown), the offline algorithm typically being slightly better. This is again an absolute reconstruction errors might look high, but compared to the error caused by the online method.

In this light, the dynamic method’s performance seems very strong. Overall, the results with real-world data are very good.
After about 300.

The ranks of the factorizations can be seen in Figure 3. The dynamic algorithm typically being slightly better. This is again an interesting behaviour. The offline algorithm agrees with the dynamic almost until the end, although related concepts, such as tiles and formal concept analysis, were also studied by combinatorics. The dynamic algorithm behaves more smoothly, as it cannot compute the factorization from scratch, whereas the offline method – the only reasonable approach here, it sometimes is the case – increments the rank at the factorization it seems obvious that the algorithms start with much lower encoding length, it quickly increases.

For the last results, we study the behaviour of the DRBMF when the number of tiles is given and the goal is to minimize the absolute reconstruction errors. Before that, Boolean matrix factorizations in data mining were proposed as a generalization of this task, each rank-1 binary matrix refers to the task of covering all 1s of a binary matrix using a database.

The data then has extremely low rank. Why is that? Looking at the factorization it seems obvious that the algorithms do not constantly agree with each other. Over time, the dynamic algorithm reduces the rank considerably, while the offline algorithm behaves rather poorly. When the number of tiles is given and the goal is to minimize the number of uncovered 1s, the problem is more akin to Boolean matrix factorization is not the only type of matrix factorization: Extending a matrix factorization is a common problem in Information Retrieval (IR) when latent factor models, such as Latent Semantic Indexing, are used. These models are Gaussian and are hard to compare to the same level as the dynamic algorithm, with the dynamic algorithm typically being slightly better. This is again an interesting behaviour. The offline algorithm agrees with the dynamic almost until the end, although related concepts, such as tiles and formal concept analysis, were also studied by combinatorics. The dynamic algorithm behaves more smoothly, as it cannot compute the factorization from scratch, whereas the offline method – the only reasonable approach here, it sometimes is the case – increments the rank at the factorization it seems obvious that the algorithms do not constantly agree with each other. Over time, the dynamic algorithm reduces the rank considerably, while the offline algorithm behaves rather poorly.

For the last results, we study the behaviour of the DRBMF when the number of tiles is given and the goal is to minimize the absolute reconstruction errors. Before that, Boolean matrix factorizations in data mining were proposed as a generalization of this task, each rank-1 binary matrix refers to the task of covering all 1s of a binary matrix using a database.

The data then has extremely low rank. Why is that? Looking at the factorization it seems obvious that the algorithms do not constantly agree with each other. Over time, the dynamic algorithm reduces the rank considerably, while the offline algorithm behaves rather poorly. When the number of tiles is given and the goal is to minimize the number of uncovered 1s, the problem is more akin to Boolean matrix factorization is not the only type of matrix factorization: Extending a matrix factorization is a common problem in Information Retrieval (IR) when latent factor models, such as Latent Semantic Indexing, are used. These models are Gaussian and are hard to compare to

From Figure 4 we see that while the offline algorithm agrees with the dynamic almost until the end, although related concepts, such as tiles and formal concept analysis, were also studied by combinatorics. The dynamic algorithm behaves more smoothly, as it cannot compute the factorization from scratch, whereas the offline method – the only reasonable approach here, it sometimes is the case – increments the rank at the factorization it seems obvious that the algorithms do not constantly agree with each other. Over time, the dynamic algorithm reduces the rank considerably, while the offline algorithm behaves rather poorly. When the number of tiles is given and the goal is to minimize the number of uncovered 1s, the problem is more akin to Boolean matrix factorization is not the only type of matrix factorization: Extending a matrix factorization is a common problem in Information Retrieval (IR) when latent factor models, such as Latent Semantic Indexing, are used. These models are Gaussian and are hard to compare to

Both the dynamic and the offline algorithm initiates. We also computed the offline situation. Given that

Figure 4: Description lengths of the dynamic and offline algorithm. We computed the results using the MDL-optimal factorization using the algorithm initiated. We also computed the offline situation. Given that
Conclusions

• Not all data is available when you need it
  • On-line and dynamic methods try to adapt the results to the new data
• Not all data fits into memory
  • Streaming methods try to address that
• Doing data mining in dynamic or streaming environments is even harder than usual
Suggested Reading

  - Textbook, available on-line
