Efficient Similarity Computation for Collaborative Filtering in Dynamic Environments

Olivier Jeunen[1], Koen Verstrepen[2], Bart Goethals[1,2]

September 18th, 2019

[1]Adrem Data Lab, University of Antwerp
[2]Froomle
olivier.jeunen@uantwerp.be

Introduction & Motivation

Setting the scene

$$\begin{pmatrix} u_1 & i_1 & t_1 \\ u_1 & i_2 & t_2 \\ u_1 & i_3 & t_3 \\ u_2 & i_4 & t_4 \\ u_2 & i_2 & t_5 \\ u_3 & i_1 & t_6 \\ u_2 & i_5 & t_7 \\ u_2 & i_7 & t_8 \\ u_3 & i_6 & t_9 \\ \ldots & \ldots & \ldots \end{pmatrix}$$

We deal with **implicit feedback**: a set of **(user, item, timestamp)**-triplets, representing clicks, views, sales, ...

Suppose we have a set of *pageviews* of this form.

Problem statement

In **neighbourhood-based** collaborative filtering[1], we need to compute **similarity** between pairs of **items**.

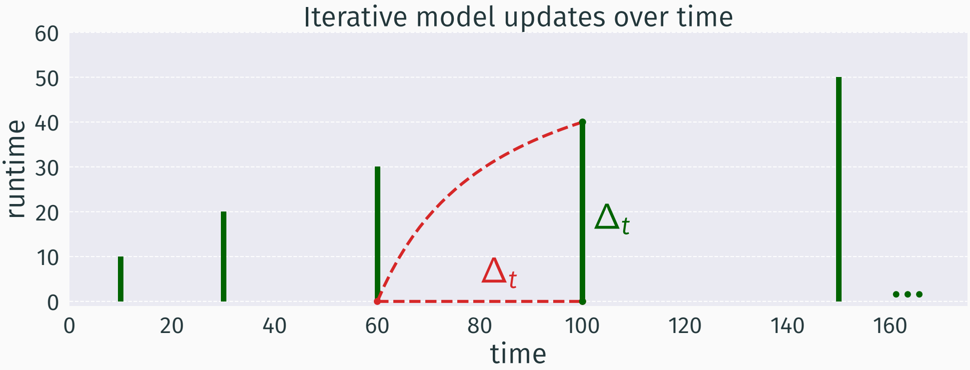
Items are represented as **sparse**, **high-dimensional** columns in the user-item matrix **P**.

$$\begin{pmatrix}
0 & 0 & 0 & \dots & 0 & 1 & 0 \\
1 & 0 & 0 & \dots & 0 & 0 & 1 \\
0 & 0 & 0 & \dots & 1 & 0 & 0 \\
0 & 0 & 1 & \dots & 0 & 0 & 0 \\
\dots & \dots & \dots & \dots & \dots & \dots & \dots \\
0 & 1 & 0 & \dots & 0 & 0 & 0 \\
0 & 0 & 0 & \dots & 0 & 1 & 0 \\
0 & 1 & 1 & \dots & 0 & 0 & 0 \\
0 & 0 & 0 & \dots & 1 & 0 & 0 \\
1 & 0 & 1 & \dots & 0 & 1 & 0
\end{pmatrix}$$

[1] Still a very competitive baseline, but often deemed unscalable

A need for speed

Typically, the model is **periodically recomputed**.
For ever-growing datasets, these **iterative** updates can become very **time-consuming** and model **recency** is often **sacrificed**.



Previous work

Existing approaches tend to **speed up** computations through

- Approximation.
- **Parallellisation**.
- **Incremental** computation.

But currently existing **exact** solutions do not **exploit** the **sparsity** that is inherent to **implicit-feedback** data streams.

Contribution & Methodology

Incremental Similarity Computation

In the binary setting, **cosine**-similarity simplifies to the number of **users** that have seen **both** items, **divided** by the square **root** of their **individual numbers**.

$$\cos(i, j) = \frac{|\mathcal{U}_i \cap \mathcal{U}_j|}{\sqrt{|\mathcal{U}_i|}\sqrt{|\mathcal{U}_j|}} = \frac{\mathsf{M}_{i,j}}{\sqrt{\mathsf{N}_i}\sqrt{\mathsf{N}_j}}$$

As such, we can compute these building **blocks incrementally** instead of recomputing the entire similarity with every update:

$$\mathsf{N} \in \mathbb{N}^n : \mathsf{N}_i = |\mathcal{U}_i| \text{ and } \mathsf{M} \in \mathbb{N}^{n \times n} : \mathsf{M}_{i,j} = |\mathcal{U}_i \cap \mathcal{U}_j|.$$

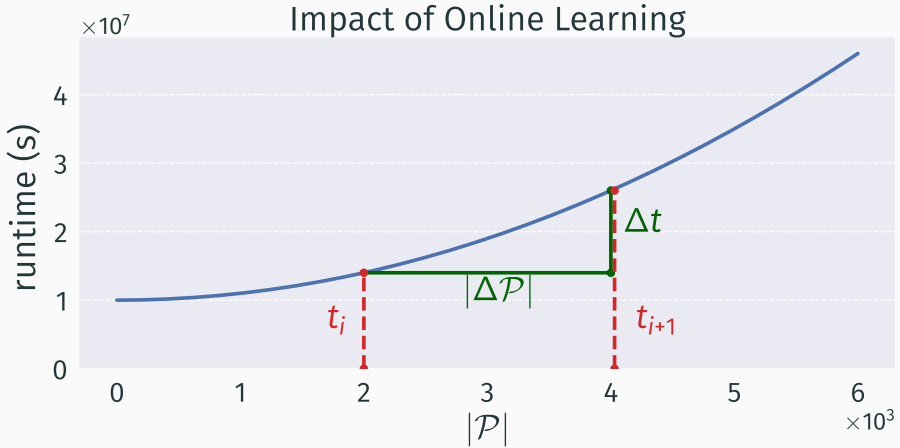
Dynamic Index

Existing approaches tend to build inverted indices in a **preprocessing** step... we do this **on-the-fly**!

Initialise a **simple inverted index for every user**, to hold their histories.
For **every pageview** (u, i):

1. **Increment** item **co-occurence** for i and **other items** seen by u.
2. **Update** the item's **count**.
3. **Add** the **item** to the user's inverted index.

As **Dynamic Index** consists of a **single for-loop** over the pageviews, it can **naturally** handle **streaming** data.



Parallellisation Procedure

We adopt a **MapReduce**-like **parallellisation** framework:

- **Mapping** is the **Dynamic Index** algorithm.
- **Reducing** two models $\mathcal{M} = \{\mathbf{M}, \mathbf{N}, \mathcal{L}\}$ and $\mathcal{M}' = \{\mathbf{M}', \mathbf{N}', \mathcal{L}'\}$ is:

  1. **Summing** up $\mathbf{M}$, $\mathbf{M}'$ and $\mathbf{N}$, $\mathbf{N}'$
  2. **Cross-referencing** (u, i)-pairs from $\mathcal{L}[u]$ with (u, j)-pairs from $\mathcal{L}'[u]$.

Step 2 is **obsolete** if $\mathcal{M}$ and $\mathcal{M}'$ are computed on **disjoint** sets of **users**!

Recommendability

Often, the set of **items** that should be considered as **recommendations** is **constrained** by recency, stock, licenses, seasonality, ... We denote $\mathcal{R}_t$ as the set of **recommendable** items at time t, and argue that it is often much **smaller** than the **full** item **collection**.

$$\|\mathcal{R}_t\| \ll \|\mathcal{I}\|$$

As such, we only need an **up-to-date** similarity $\text{sim}(i, j)$ if either i or j is **recommendable**:

$$i \in \mathcal{R}_t \vee j \in \mathcal{R}_t$$

To keep **up-to-date** with recommendability updates:
add a **second inverted index** for every user.

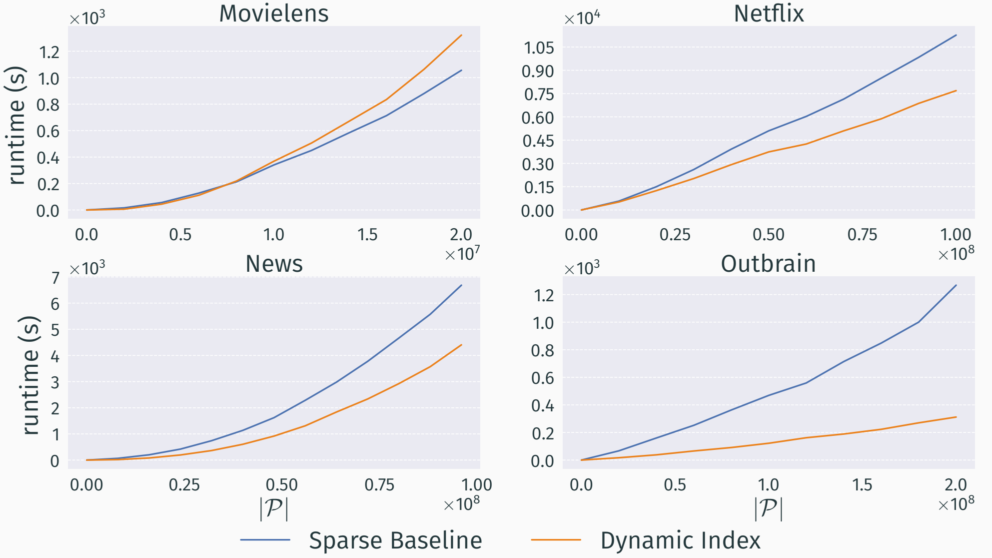
Experimental Results

Datasets

Table 1: Experimental dataset characteristics.

	Movielens*	Netflix*	News	Outbrain
# “events”	20e6	100e6	96e6	**200e6**
# users	138e3	480e3	5e6	**113e6**
# items	27e3	18e3	297e3	**1e6**
mean items per user	144.41	**209.25**	18.29	1.76
mean users per item	747.84	**5654.50**	242.51	184.50
sparsity user-item matrix	99.46%	98.82%	**99.99%**	**99.99%**
sparsity item-item matrix	59.90%	0.22%	99.83%	**99.98%**

RQ1: Are we more efficient than the baselines?

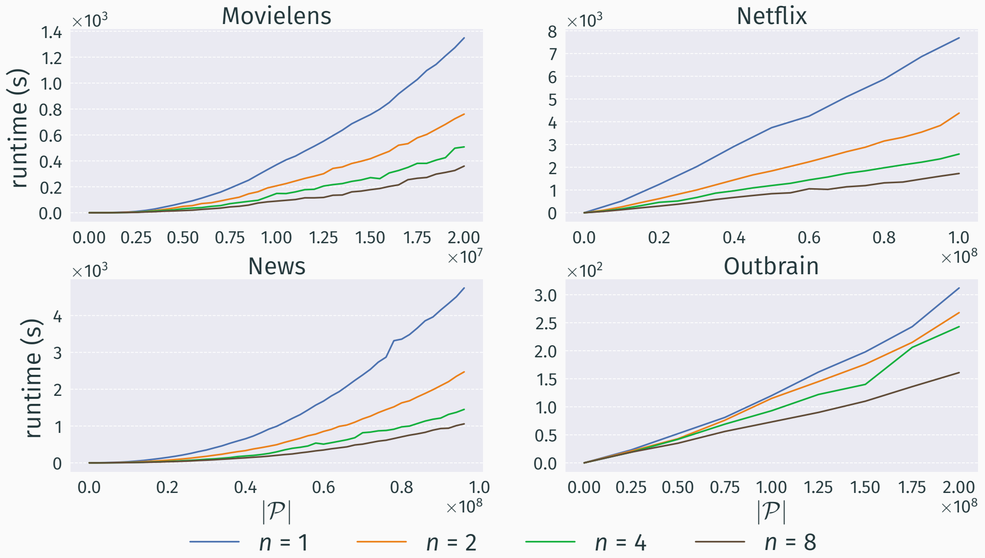


RQ1: Are we more efficient than the baselines?

Observations

- More **efficient** if **M** is **sparse**
- More **efficient** if users have **shorter histories**
- Average number of processed **interactions per second** ranges from **14 500** to **834 000**

RQ2: How effective is parallellisation?

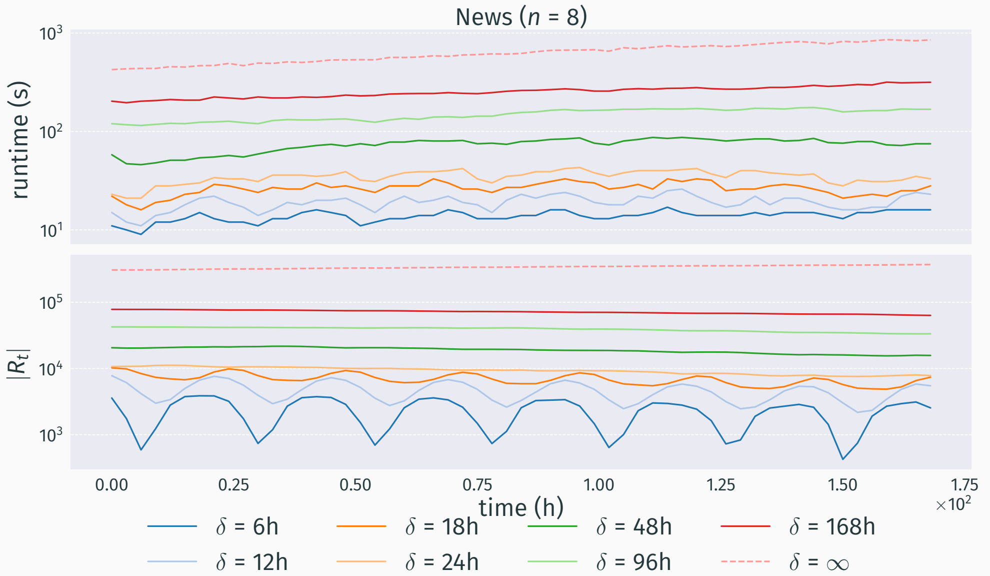


RQ2: How effective is parallellisation?

Observations

- **Speedup** factor of $> \mathbf{4}$ for Netflix and News datasets with **8 cores**
- **Incremental** updates **complicate reduce** procedure:
  - For sufficiently **large batches**, performance **gains** are **tangible**.
  - For **small batches**, **single-core** updates are **preferred**.

RQ3: What is the effect of constrained recommendability?



RQ3: What is the effect of constrained recommendability?

Observations

- Clear **efficiency gains** for lower values of δ:
  - **48h** only needs $<$ **10%** of the runtime needed without restrictions.
  - **24h** $<$ **5%**
  - **6h** **1.6%**
- **Slope** of increasing runtime with more data is **flattened**, improving **scalability**.

Conclusion & Future Work

Conclusion

We introduce **Dynamic Index**, which:

- is **faster** than the state-of-the art in **exact similarity computation** for **sparse** and **high-dimensional** data.

Conclusion

We introduce **Dynamic Index**, which:

- is **faster** than the state-of-the art in **exact similarity computation** for **sparse** and **high-dimensional** data.
- computes **incrementa**lly by **design**.

Conclusion

We introduce **Dynamic Index**, which:

- is **faster** than the state-of-the art in **exact similarity computation** for **sparse** and **high-dimensional** data.
- computes **incrementally** by **design**.
- is easily **parallellisable**.

Conclusion

We introduce **Dynamic Index**, which:

- is **faster** than the state-of-the art in **exact similarity computation** for **sparse** and **high-dimensional** data.
- computes **incrementally** by **design**.
- is easily **parallellisable**.
- naturally handles and **exploits recommendability** of items.

Questions?

Source code is available:

Academics hire too!

PhD students + Post-docs

Adrem

Adrem Data Lab

University of Antwerp

Future Work

- More advanced similarity **measures**:
  - **Jaccard** index, Pointwise Mutual Information (**PMI**), **Pearson** correlation,...
    are all dependent on the **co-occurrence matrix M**.

- **Beyond item-item** collaborative filtering:
  - With relatively straightforward **extensions**...
    (e.g. including a value in the inverted indices to allow for **non-binary data**)
    ...we can tackle more **general Information Retrieval** use-cases.