Applied ML at Scale

Ulrich Paquet

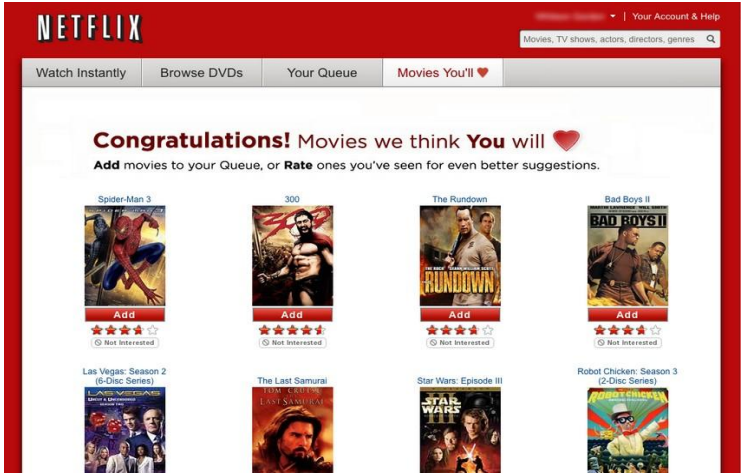

I MADE THIS
LAUNCH TEAM 2013

The early days

The story of Greg Linden and Amazon recommendations

Netflix - Predicting Ratings

Long Ago… The Netflix Prize



The goal: 10% improvement in RMSE over Netflix's Cinematch

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

It took tens of thousands of participants over 2 years….

Netflix Prize – The Rules

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- A trivial “mean-score” prediction would yield a RMSE = 1.0540
- Cinematch yielded a RMSE = 0.9514
- A 10% improvement to win the grand prize meant a RMSE ≤ 0.8572

Netflix Prize Timeline

- **October 2, 2006** – Competition started
  - After 6 days: Cinematch was beaten
  - After 1 week: First team qualified for the annual progress prize (1% over Cinematch)
  - After 1 year: Over 40,000 teams from 186 countries have joined

Netflix Prize Timeline

- **October 2, 2006** – Competition started
  - After 6 days: Cinematch was beaten
  - After 1 week: First team qualified for the annual progress prize (1% over Cinematch)
  - After 1 year: Over 40,000 teams from 186 countries have joined

- **November 13, 2007** –Yehuda Koren, Robert Bell and Chris Volinsky from AT&T (*BellKor*) received the $50,000 progress prize with RMSE=0.8712 (8.43% improvement)

Netflix Prize Timeline

- **October 2, 2006** – Competition started
  - After 6 days: Cinematch was beaten
  - After 1 week: First team qualified for the annual progress prize (1% over Cinematch)
  - After 1 year: Over 40,000 teams from 186 countries have joined

- **November 13, 2007** –Yehuda Koren, Robert Bell and Chris Volinsky from AT&T (*BellKor*) received the $50,000 progress prize with RMSE=0.8712 (8.43% improvement)

- The 2008 progress prize was given to a joint solution by *BellKor* and *BigChaos* (Andreas Toscher and Michael Jahrer). RMSE=0.8616 and an assemble of 207 predictors sets.

- **June 26, 2009** – A merger of *BellKor*, *BigChaos* and *Pragmatic Theory* achieved a 10.05% improvement over Cinematch. The competition entered a 30-days “last call” period

- **July 26, 2009** – The competition stopped gathering submissions.
  The leaderboard showed:
  1. *The Ensemble* with 10.10% improvement
  2. *BellKor’s Pragmatic Chaos* with a 10.9% improvement

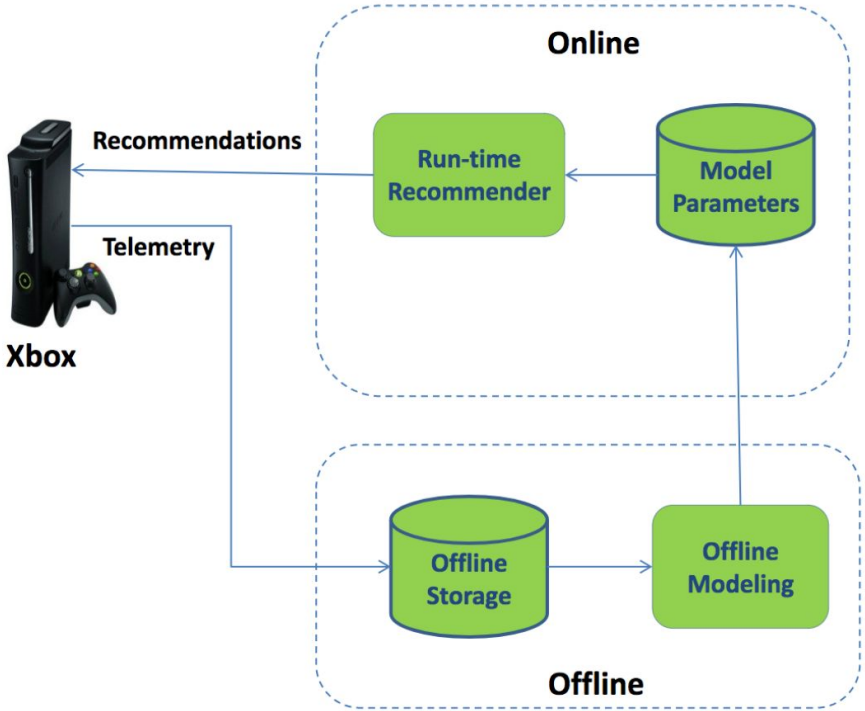
- **September 18, 2009** – Netflix announced *BellKor’s Pragmatic Chaos* as the winner of the competition.
  Their results matched those of *The Ensemble* but since they submitted 20 minutes later, the rules awarded the prize to *BellKor*.

Yehuda Koren's Matrix Factorization Techniques for Recommender Systems

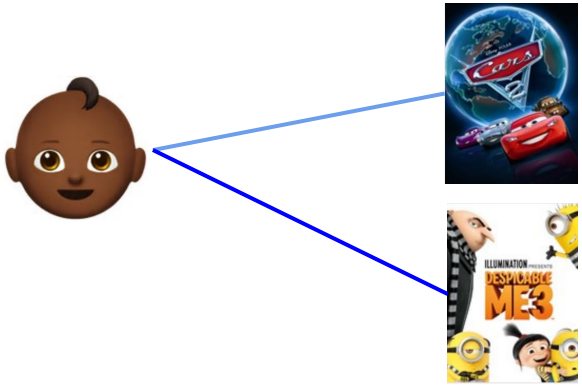
Step by step: Yahoo! Music Recommendations: Modeling Music Ratings with Temporal Dynamics and Item Taxonomy

The Xbox Recommender System

A new component in Xbox 360



The goal



You may also like

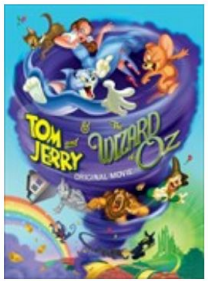
Rio

The Smurfs

Hop

Kung Fu Panda 2

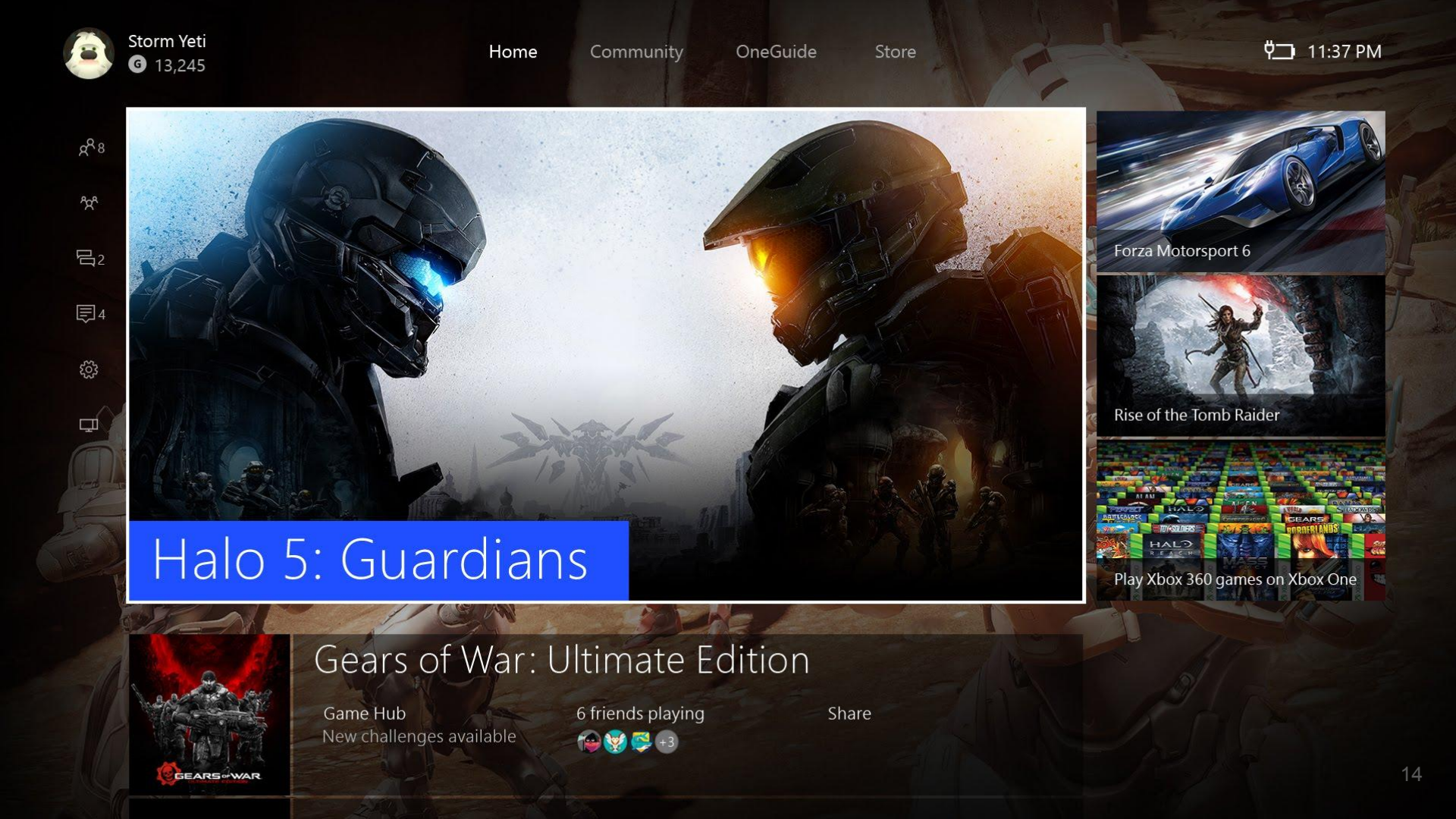
Scooby-Doo! Leg-
end of the Phan-



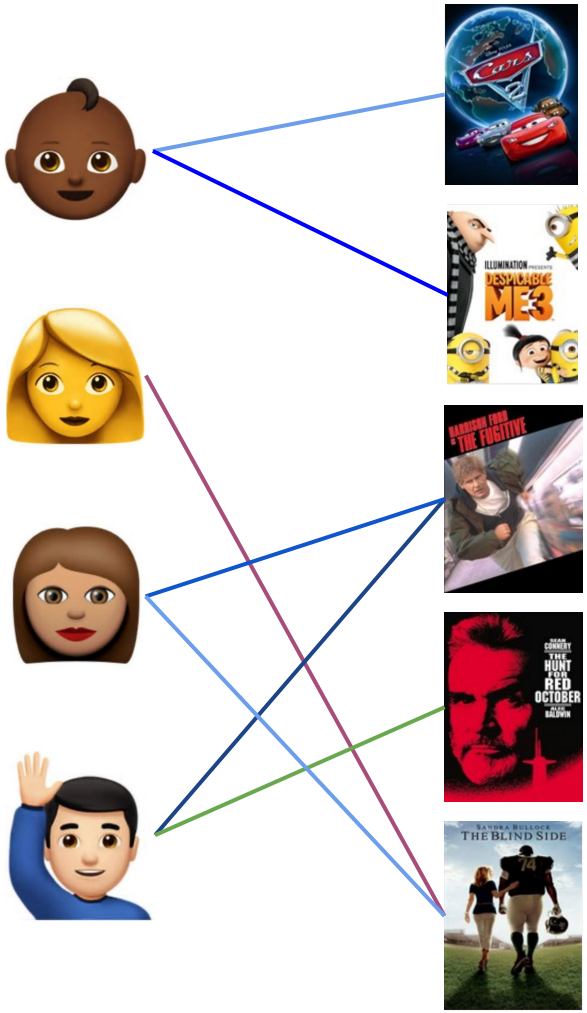
Tom and Jerry &
The Wizard of Oz

Happy Feet Two

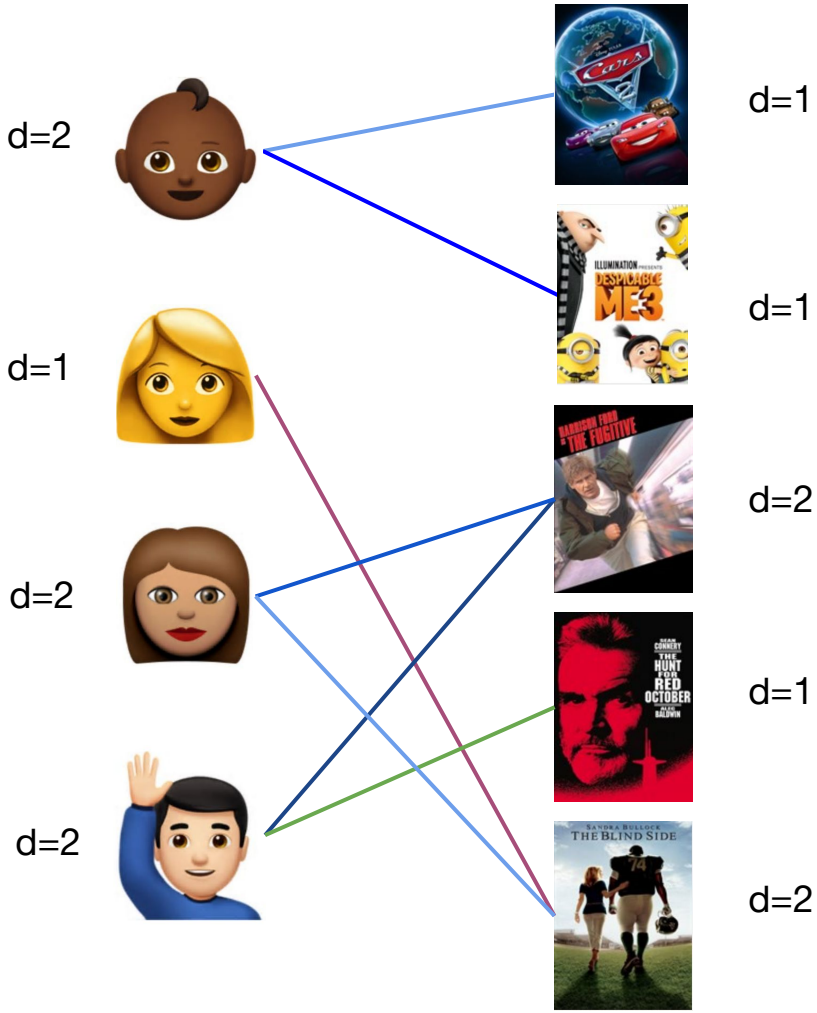
Gnomeo & Juliet


Storm Yeti
13,245
Home
Community
OneGuide
Store
11:37 PM
8
2
4
Halo 5: Guardians
Forza Motorsport 6
Rise of the Tomb Raider
Play Xbox 360 games on Xbox One
Gears of War: Ultimate Edition
Game Hub
New challenges available
6 friends playing
+3
Share
14

The kind of data

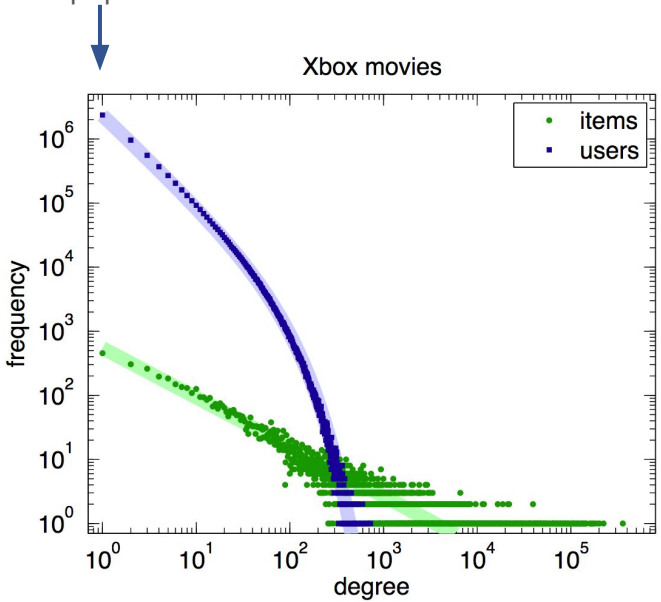


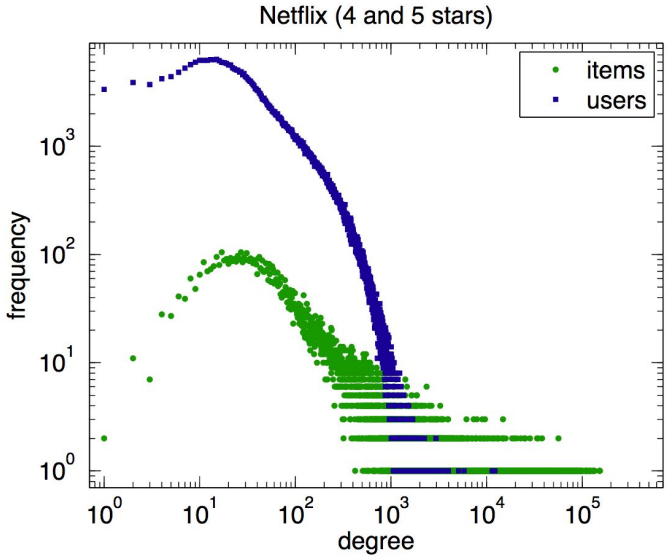
Degree distributions



Power laws

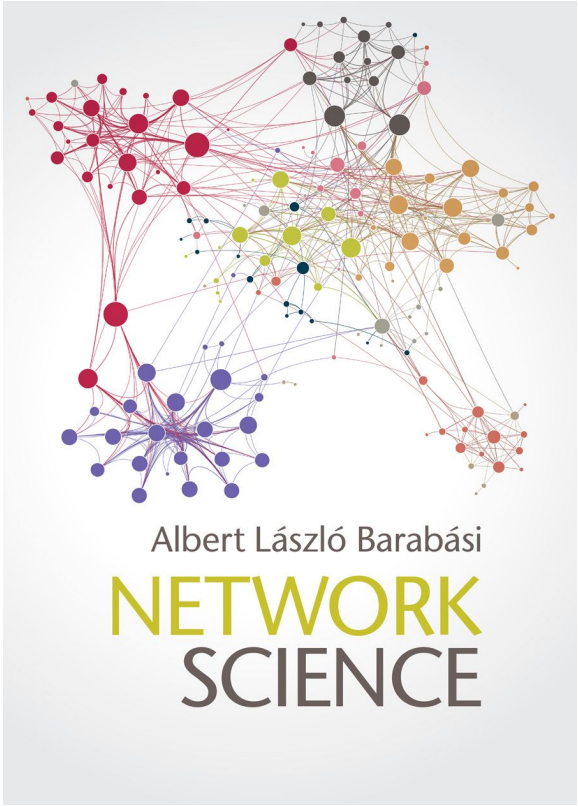
The least popular movie





$$p_{\text{item}}(d) \propto d^{-0.77}$$
$$p_{\text{user}}(d) \propto d^{-1.4}\, \mathrm{e}^{-d/70}$$

A good read



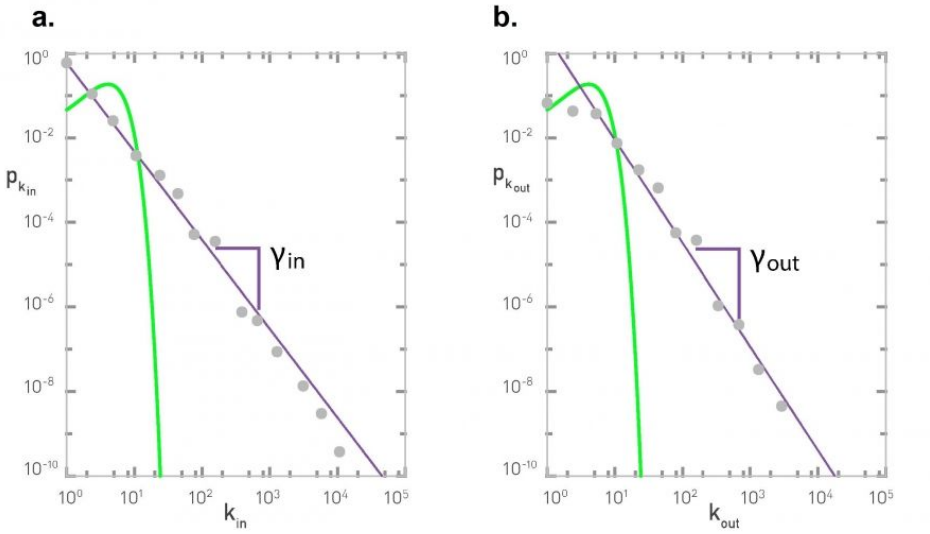
http://networksciencebook.com/

Power laws and scale free networks

The probability p_k that a node has exactly k links (degree) is:

$$p_k \sim k^{-\gamma}$$

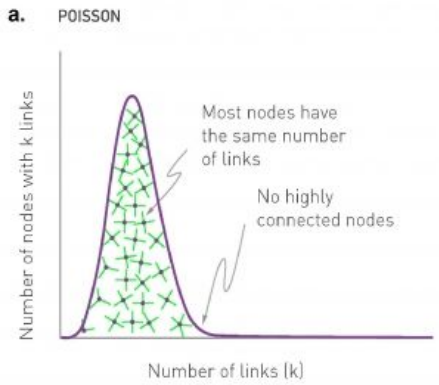
$$\log p_k \sim -\gamma \log k$$

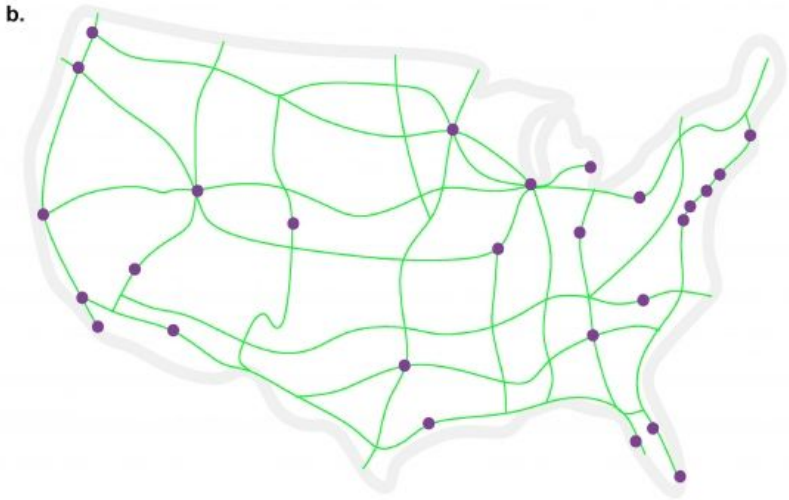


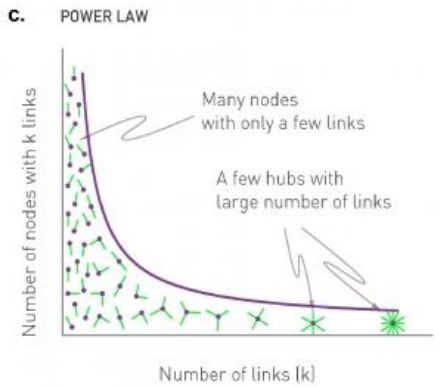
The Degree Distribution of the WWW
The incoming (a) and outgoing (b) degree distribution of the WWW sample mapped in the 1999 study of Albert *et al*. [1]. The degree distribution is shown on double logarithmic axis (log-log plot), in which a power law follows a straight line. The symbols correspond to the empirical data and the line corresponds to the power-law fit, with degree exponents γ_{in}= 2.1 and γ_{out} = 2.45. We also show as a green line the degree distribution predicted by a Poisson function with the average degree $\langle k_{in} \rangle = \langle k_{out} \rangle$ = 4.60 of the WWW sample.

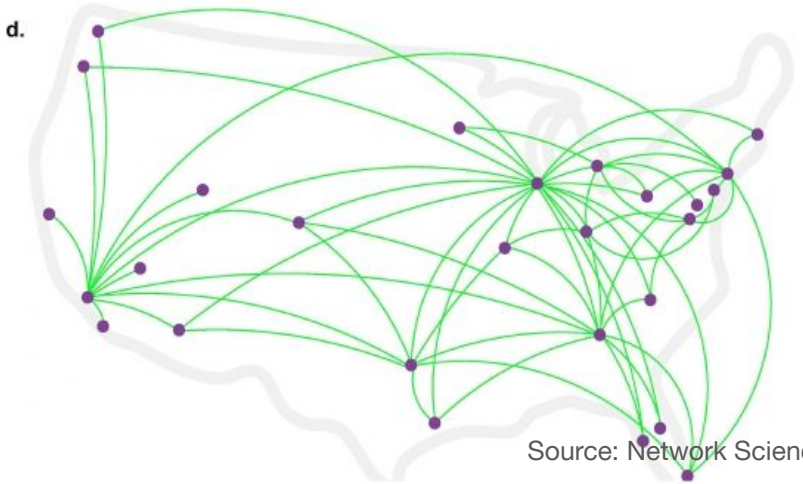
Poisson → $p_k = e^{-\langle k \rangle} \frac{\langle k \rangle}{k!}$

Source: Network Science, Albert-László Barabási









$$p_k = \frac{1}{\sum_{j=1}^{\infty} j^{-\gamma}} k^{-\gamma} = \frac{1}{\zeta(\gamma)} k^{-\gamma}$$

The probability p_k that a node has exactly k links; special case for p_0.

Source: Network Science, Albert-László Barabási

Power laws and scale free networks

$$\langle k^n \rangle = \int_{k_{\min}}^{k_{\max}} k^n \, p(k) \, \mathrm{d}k$$
$$= \frac{1}{Z} \int_{k_{\min}}^{k_{\max}} k^n \, k^{-\gamma} \, \mathrm{d}k$$
$$= \frac{1}{Z} \left(k_{\max}^{n-\gamma+1} - k_{\min}^{n-\gamma+1} \right) \Big/ (n - \gamma + 1)$$

The degree of the largest hub, k_{max}, increases with the system size

To understand the behavior of $\langle k^n \rangle$ we need to take the asymptotic limit $k_{max} \to \infty$.

All moments larger than $\gamma-1$ (i.e. $n-\gamma+1 > 0$) diverge

Lack of an Internal Scale

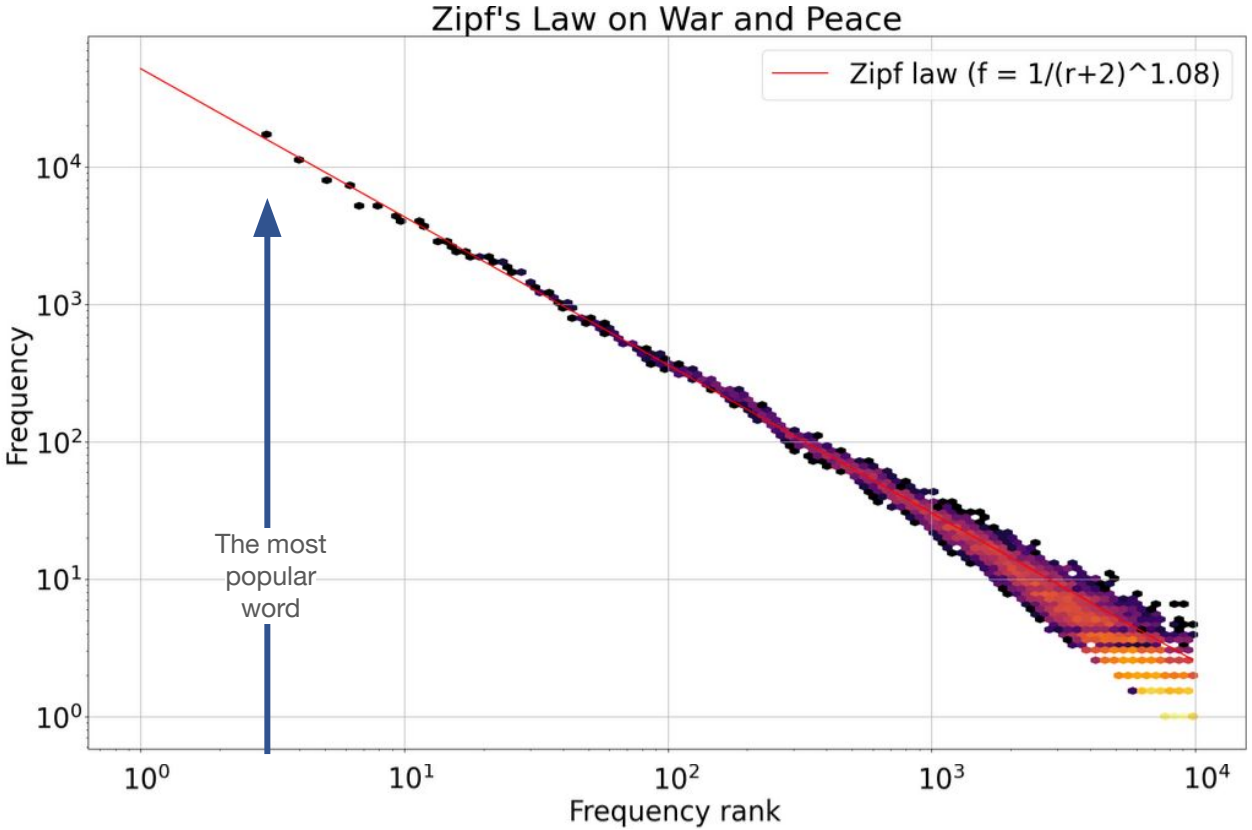
For any exponentially bounded distribution, like a Poisson or a Gaussian, the degree of a randomly chosen node is in the vicinity of $\langle k \rangle$. Hence $\langle k \rangle$ serves as the network's *scale*.

For a power law distribution the second moment can diverge, and the degree of a randomly chosen node can be significantly different from $\langle k \rangle$. Hence $\langle k \rangle$ does not serve as an intrinsic scale. A network with a power law degree distribution lacks an intrinsic scale…

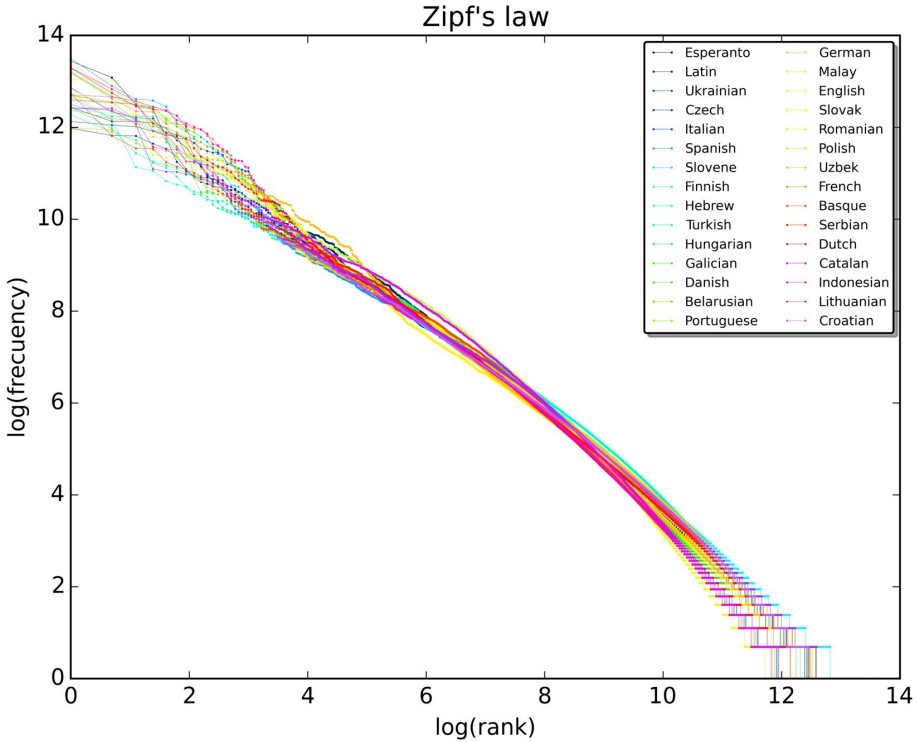
Source: Network Science, Albert-László Barabási

Zipf’s law

$$\text{word frequency} \propto \frac{1}{\text{word rank}}$$

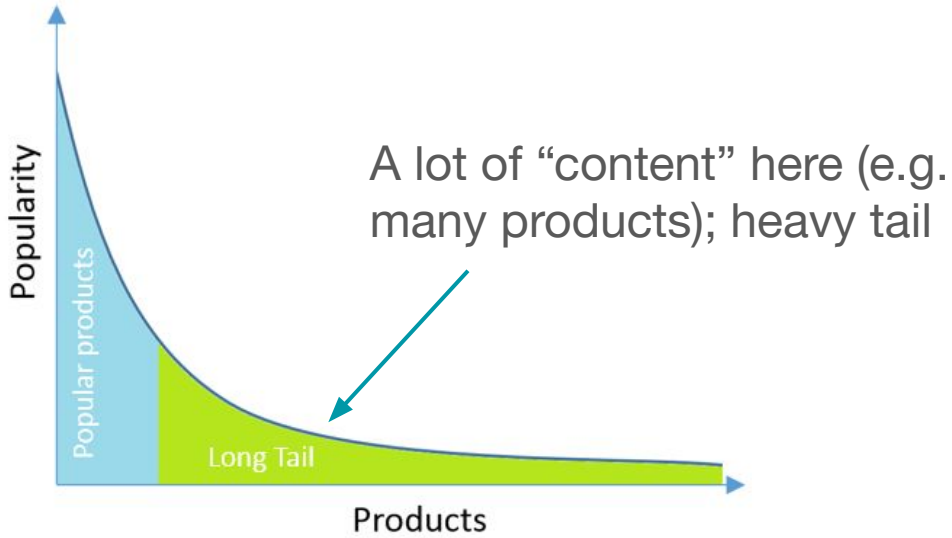


Zipf’s law



Zipf's law plot for the first 10 million words in 30 Wikipedias (as of October 2015) in a log-log scale.

Power law distributions in online data



Approaches

[adventure, animated, boys, cars, kids]
[0 1 1 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0]

[adventure, animated, anti-hero, kids]
[0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

1. Feature-based
2. Nearest neighbour-based
3. Collaborative filtering

   and then blending them

4. Modern approaches based on reinforcement learning, maximizing an expected “future reward”

Approaches

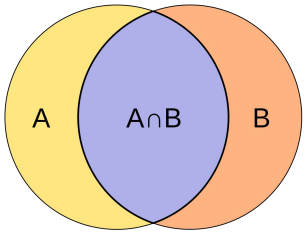
1. Feature-based

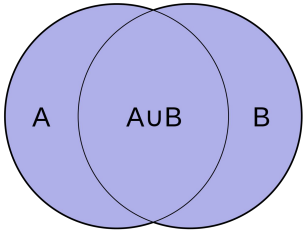
   Jaccard similarity

[adventure, animated, boys, cars, kids]
[0 1 1 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0]

[adventure, animated, anti-hero, kids]
[0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

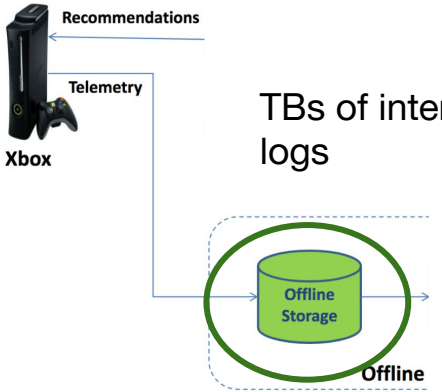




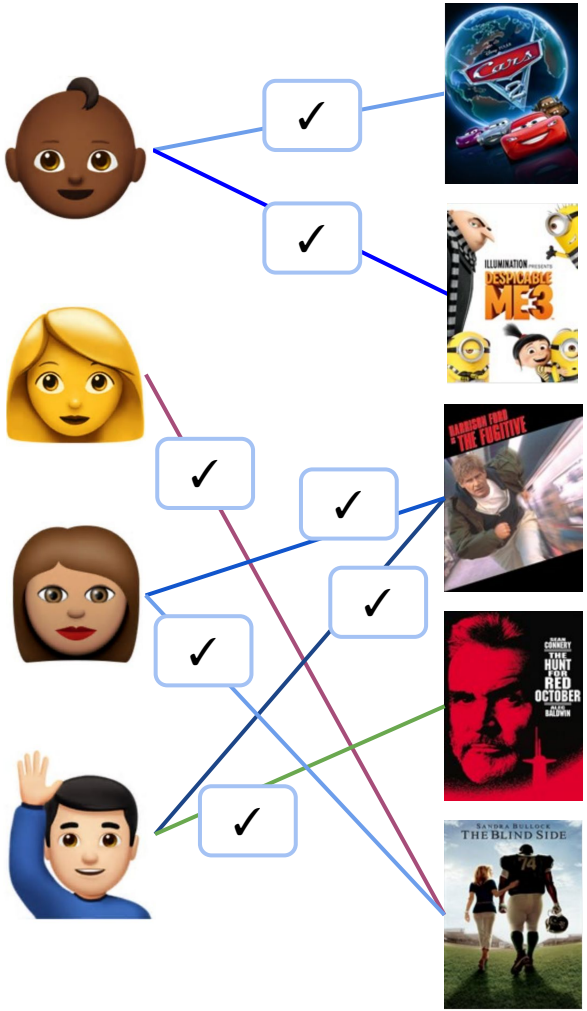
Xbox movies and games

2010, 2011

The early days



TBs of interaction logs



The Netflix Prize

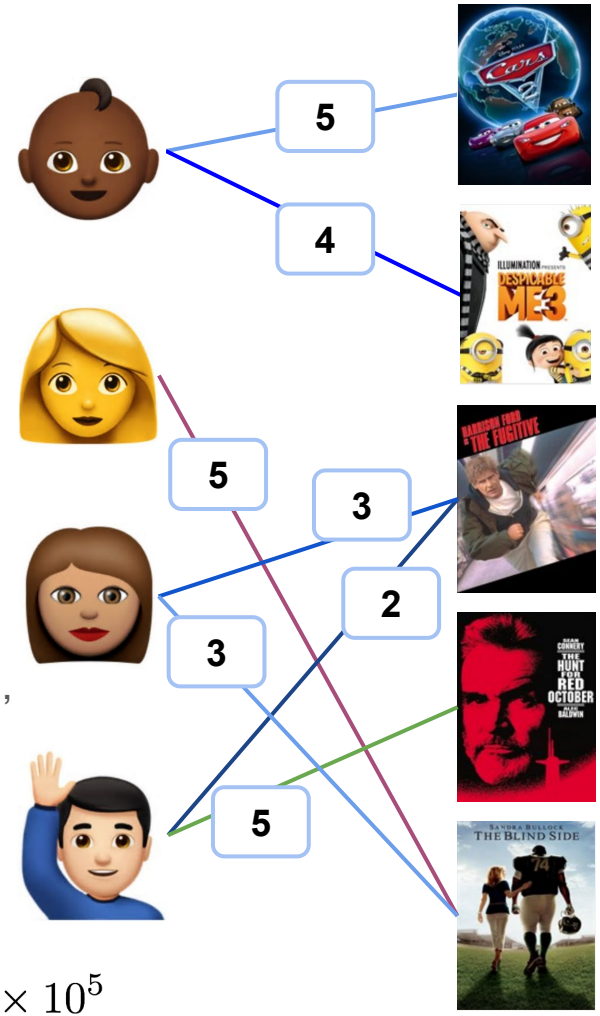
2007

The really early days

$1M prize money for best test RMSE :)

1x solution per 24 hours

Winning entries not a single model, but ensemble of many models...

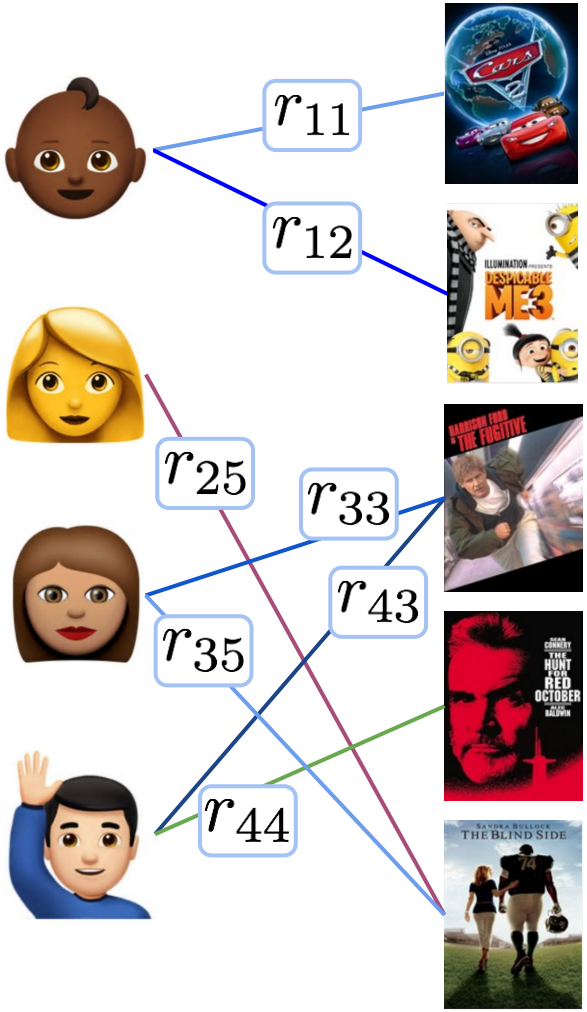


5 ★★★★★

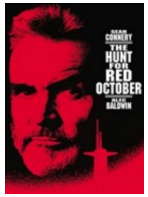
10^8 ratings

$M = 4.8 \times 10^5$

$N = 1.8 \times 10^4$


r_{11}
r_{12}
r_{25}
r_{33}
r_{43}
r_{35}
r_{44}
CARS 2
DESPICABLE ME 3
THE FUGITIVE
THE HUNT FOR RED OCTOBER
THE BLIND SIDE



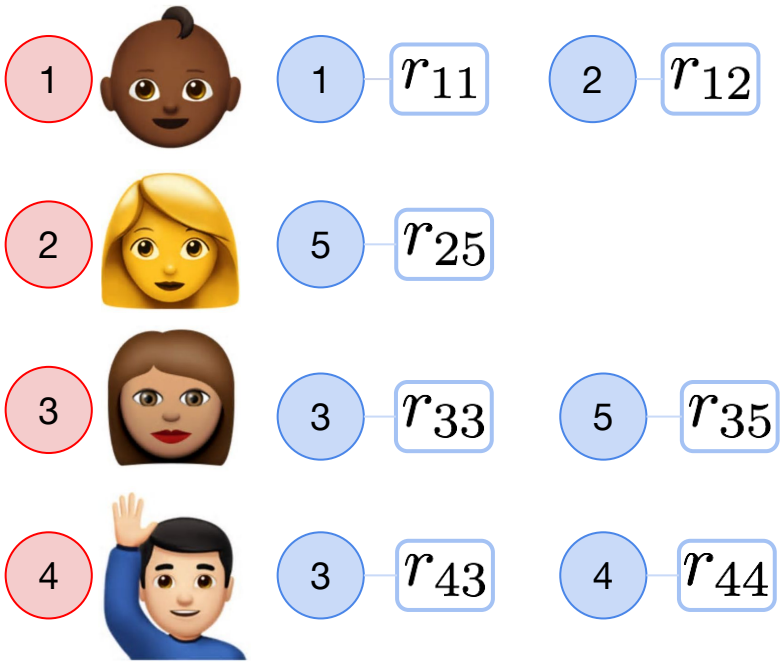

r_{11} r_{12}

r_{25}

r_{33} r_{35}

r_{43} r_{44}

Indexing twice



Practical 1

Understanding the data and the problem

Practical 1 | **Reading list**

Read:

✅ Yehuda Koren's Matrix Factorization Techniques for Recommender Systems

✅ Step by step: Yahoo! Music Recommendations: Modeling Music Ratings with Temporal Dynamics and Item Taxonomy

✅ The Xbox Recommender System

Practical 1

✅ Download the MovieLens 32M data set and understand its README

Dataset examples

movies.csv

```
movieId,title,genres
1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy
2,Jumanji (1995),Adventure|Children|Fantasy
3,Grumpier Old Men (1995),Comedy|Romance
```

tags.csv

```
userId,movieId,tag,timestamp
3,260,classic,1439472355
3,260,sci-fi,1439472256
4,1732,dark comedy,1573943598
4,1732,great dialogue,1573943604
```

ratings.csv

```
userId,movieId,rating,timestamp
1,296,5.0,1147880044
1,306,3.5,1147868817
```

Practical 1

✅ Download the MovieLens 32M data set and understand its README

✅ Write a data structure that **indexes** the data (a) by user and (b) by movie

✅ Plot rating distributions of data

✅ Are there power laws in the data? Is it scale-free? (Note that each user rated at least 20 movies. Truncation will skew the power law that you'd expect…)

$b_1^{(u)}$ $\mathbf{u}_1$

$b_2^{(u)}$ $\mathbf{u}_2$

$b_3^{(u)}$ $\mathbf{u}_3$

$b_4^{(u)}$ $\mathbf{u}_4$

r_{11}

r_{12}

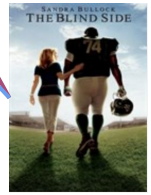
r_{25}

r_{33}

r_{43}

r_{35}

r_{44}



$\mathbf{v}_1$ $b_1^{(i)}$

$\mathbf{v}_2$ $b_2^{(i)}$

$\mathbf{v}_3$ $b_3^{(i)}$

$\mathbf{v}_4$ $b_4^{(i)}$

$\mathbf{v}_5$ $b_5^{(i)}$

$b_1^{(u)}$ $\mathbf{u}_1$

$b_2^{(u)}$ $\mathbf{u}_2$

$b_3^{(u)}$ $\mathbf{u}_3$

$b_4^{(u)}$ $\mathbf{u}_4$

r_{11}

r_{12}

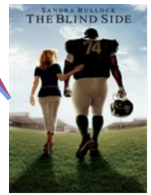
r_{25}

r_{33}

r_{43}

r_{35}

r_{44}



$\mathbf{v}_1$ $b_1^{(i)}$

$\mathbf{v}_2$ $b_2^{(i)}$

$\mathbf{v}_3$ $b_3^{(i)}$

$\mathbf{v}_4$ $b_4^{(i)}$

$\mathbf{v}_5$ $b_5^{(i)}$

$b_1^{(u)}$ $\mathbf{u}_1$

$b_2^{(u)}$ $\mathbf{u}_2$

$b_3^{(u)}$ $\mathbf{u}_3$

$b_4^{(u)}$ $\mathbf{u}_4$

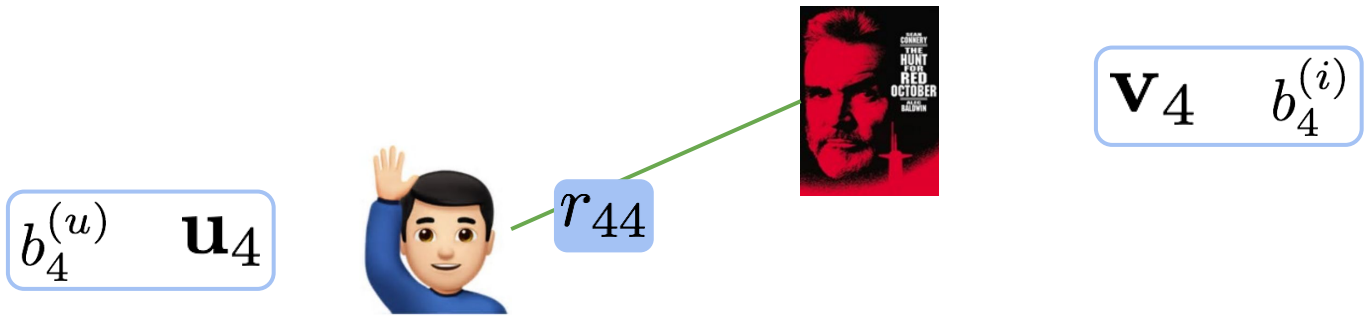
In neural network terms:

Each user is represented as a one hot vector (millions of dimensions big, though)

Each user is embedded as a low dimensional vector

A simple likelihood

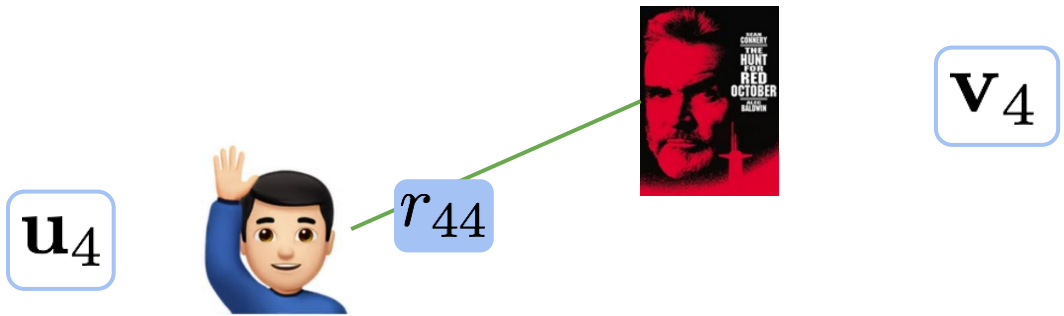
$$p(r_{mn}|\mathbf{u}_m, \mathbf{v}_n, b_m^{(u)}, b_n^{(i)}) = \mathcal{N}(r_{mn}\,;\, \mathbf{u}_m^T\mathbf{v}_n + b_m^{(u)} + b_n^{(i)},\, \lambda^{-1})$$



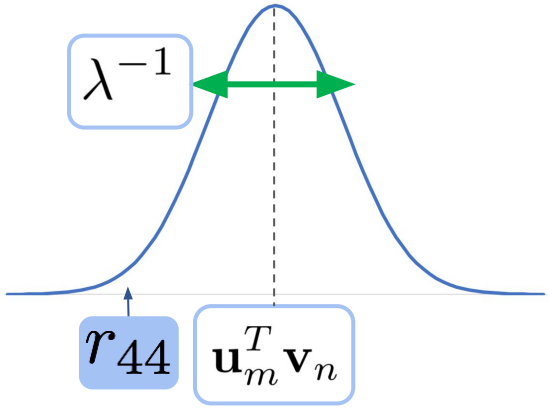
A simple likelihood

We'll remove biases to make the storyline simpler ;-)

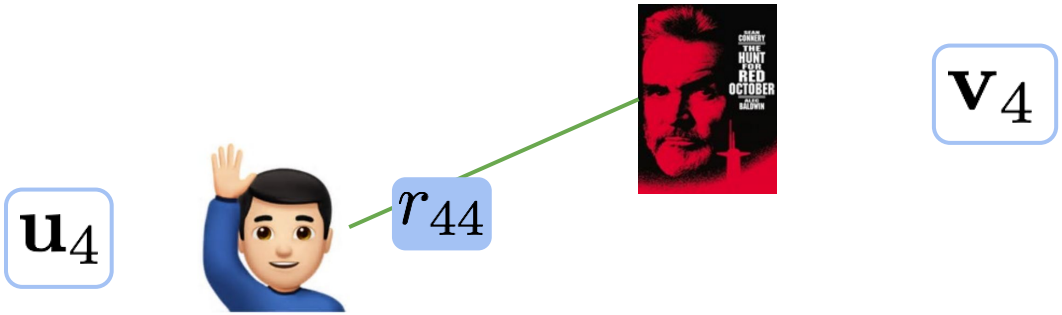
$$p(r_{mn}|\mathbf{u}_m, \mathbf{v}_n) = \mathcal{N}(r_{mn}\,;\, \mathbf{u}_m^T\mathbf{v}_n,\, \lambda^{-1})$$



A simple likelihood

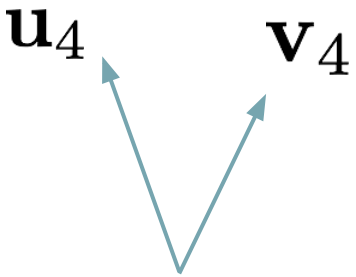


$$p(r_{mn}|\mathbf{u}_m, \mathbf{v}_n) = \mathcal{N}(r_{mn}\,;\, \mathbf{u}_m^T \mathbf{v}_n,\, \lambda^{-1})$$

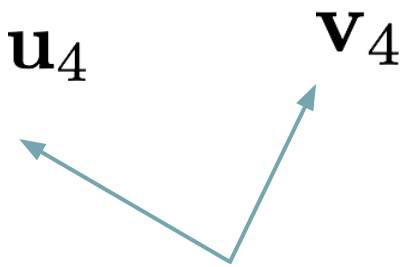


A simple likelihood

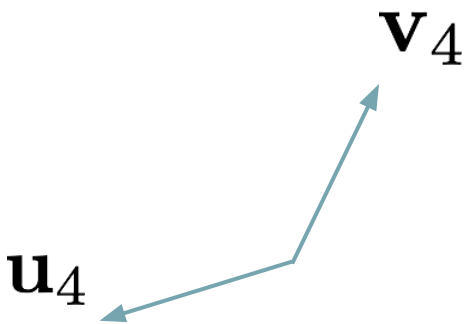
$$p(r_{mn}|\mathbf{u}_m, \mathbf{v}_n) = \mathcal{N}(r_{mn}\,;\, \mathbf{u}_m^T\mathbf{v}_n,\, \lambda^{-1})$$



(large) positive r_{44}



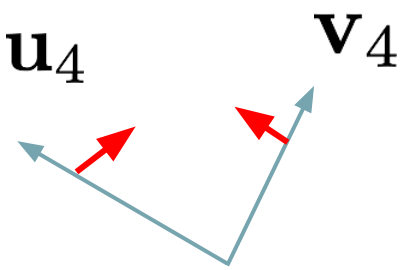
zero r_{44}



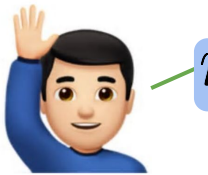
(large) negative r_{44}

What should happen during optimization?

$$p(r_{mn}|\mathbf{u}_m, \mathbf{v}_n) = \mathcal{N}(r_{mn}\,;\, \mathbf{u}_m^T \mathbf{v}_n,\, \lambda^{-1})$$



positive r_{44}

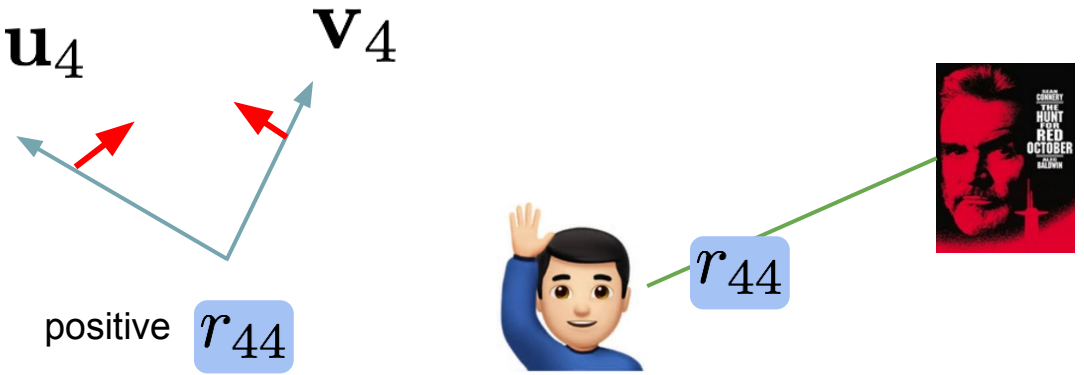


r_{44}

A great solution...

Embeds all users and items, so that

- The inner product between similar users / items is big: they are embedded in the same part of the space
- The inner product between different users / items is negative: they are embedded in a different part of the space.



A log likelihood / objective function

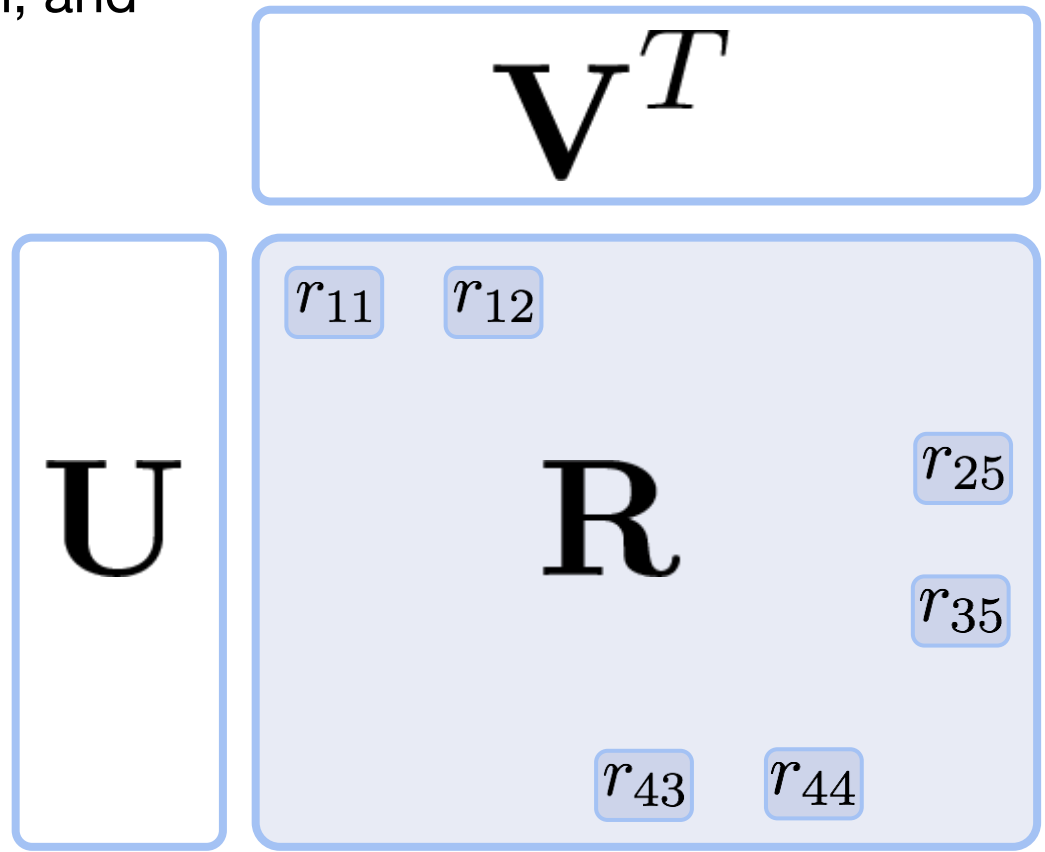
$$\log p(\mathbf{R}|\mathbf{U},\mathbf{V}) = -\frac{\lambda}{2}\sum_{m=1}^{M}\sum_{n\in\Omega(m)}(r_{mn}-\mathbf{u}_m^T\mathbf{v}_n)^2 + \text{const}$$

$$= -\frac{\lambda}{2}\sum_{n=1}^{N}\sum_{m\in\Pi(n)}(r_{mn}-\mathbf{u}_m^T\mathbf{v}_n)^2 + \text{const}$$

summing over all observed pairs

Matrix factorization, and a low-rank approximation

$$\mathbf{R} \approx \mathbf{U}\mathbf{V}^T$$



A regularized log likelihood

$$\begin{aligned}\mathcal{L} &= \log p(\mathbf{R}|\mathbf{U},\mathbf{V}) + \log p(\mathbf{U}) + \log p(\mathbf{V}) \\ &= -\frac{\lambda}{2}\sum_{mn}(r_{mn} - \mathbf{u}_m^T\mathbf{v}_n)^2 \\ &\quad -\frac{\tau}{2}\sum_m \mathbf{u}_m^T\mathbf{u}_m - \frac{\tau}{2}\sum_n \mathbf{v}_n^T\mathbf{v}_n + \text{const}\end{aligned}$$

Gradient descent on *L*

$$n \in \Omega(m)$$

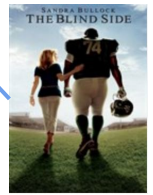
$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}_m} = \mathbf{0}$$

$\mathbf{u}_3$

r_{33}

r_{35}

$\mathbf{v}_3$



$\mathbf{v}_5$

Gradient descent on *L*

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}_m} = \mathbf{0}$$

$$\mathbf{u}_m = \left(\lambda \sum_{n \in \Omega(m)} \mathbf{v}_n \mathbf{v}_n^T + \tau \mathbf{I} \right)^{-1} \left(\lambda \sum_{n \in \Omega(m)} r_{mn} \mathbf{v}_n \right)$$

Gradient descent on *L*

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}_m} = \mathbf{0}$$

$$\mathbf{u}_m = \left(\lambda \sum_{n \in \Omega(m)} \mathbf{v}_n \mathbf{v}_n^T + \tau \mathbf{I} \right)^{-1} \left(\lambda \sum_{n \in \Omega(m)} r_{mn} \mathbf{v}_n \right)$$

Can you derive this? 5 minute class exercise.

Derivatives cheat sheet

$$\frac{\partial(\mathbf{a}^T\mathbf{x})}{\partial\mathbf{x}} = \frac{\partial(\mathbf{x}^T\mathbf{a})}{\partial\mathbf{x}} = \mathbf{a}$$

$$\frac{\partial(\mathbf{x}^T\mathbf{A}\mathbf{x})}{\partial\mathbf{x}} = (\mathbf{A} + \mathbf{A}^T)\mathbf{x}$$

$$\frac{\partial(\mathbf{x}^T\mathbf{A})}{\partial\mathbf{x}} = \mathbf{A}$$

scratch space for lectures

$$\mathcal{L} = -\frac{\lambda}{2}\sum_{mn}(r_{mn} - \mathbf{u}_m^T\mathbf{v}_n)^2 - \frac{\tau}{2}\sum_m \mathbf{u}_m^T\mathbf{u}_m + \cdots$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}_m} = -\lambda \sum_{n\in\Omega(m)} -\mathbf{v}_n(r_{mn} - \mathbf{v}_n^T\mathbf{u}_m) - \tau\mathbf{u}_m$$

$$= 0$$

$$\mathcal{L} = -\frac{\lambda}{2}\sum_{mn}(r_{mn} - \mathbf{u}_m^T\mathbf{v}_n)^2 - \frac{\tau}{2}\sum_m \mathbf{u}_m^T\mathbf{u}_m + \cdots$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}_m} = -\lambda \sum_{n\in\Omega(m)} -\mathbf{v}_n(r_{mn} - \mathbf{v}_n^T\mathbf{u}_m) - \tau\mathbf{u}_m$$

$$= 0$$

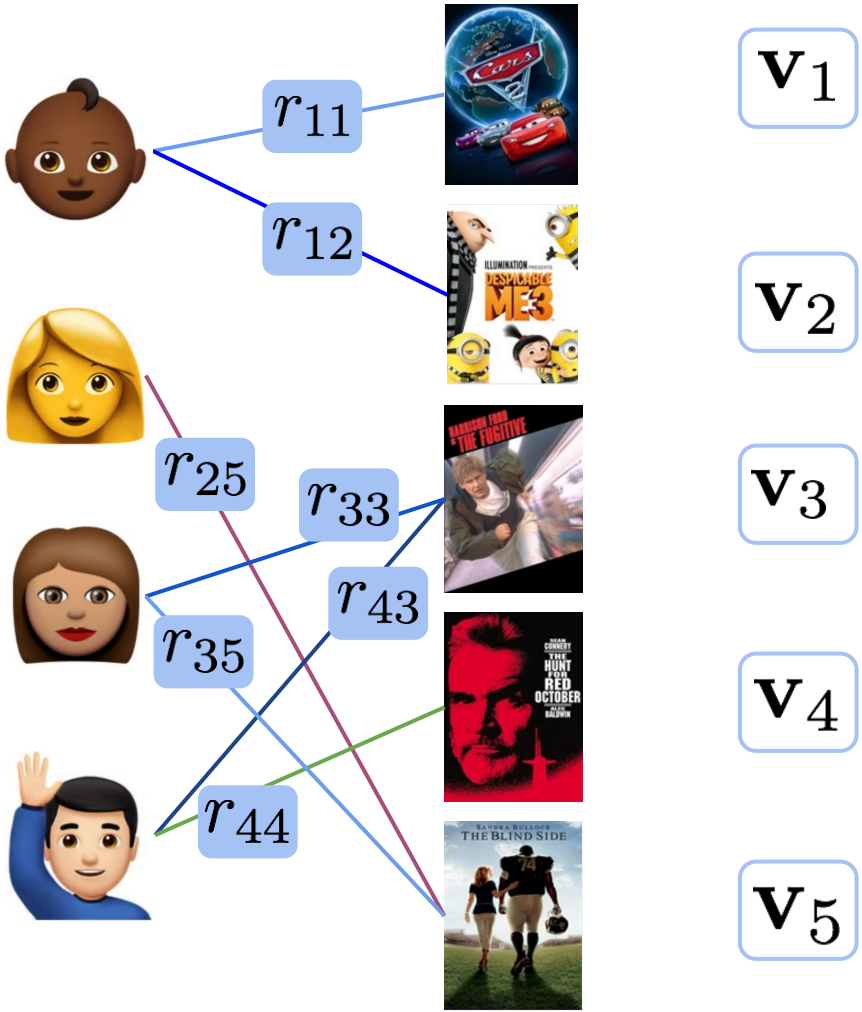
$$\left(\lambda \sum_{n\in\Omega(m)} \mathbf{v}_n\mathbf{v}_n^T + \tau\mathbf{I}\right)\mathbf{u}_m = \lambda \sum_{n\in\Omega(m)} \mathbf{v}_n r_{mn}$$

$$\mathbf{u}_m = \left(\lambda \sum_{n\in\Omega(m)} \mathbf{v}_n\mathbf{v}_n^T + \tau\mathbf{I}\right)^{-1}\left(\lambda \sum_{n\in\Omega(m)} \mathbf{v}_n r_{mn}\right)$$

Gradient descent on *L*

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}_m} = \mathbf{0}$$

$\mathbf{u}_3$



$$\mathbf{u}_m = \left(\lambda \sum_{n \in \Omega(m)} \mathbf{v}_n \mathbf{v}_n^T + \tau \mathbf{I} \right)^{-1} \left(\lambda \sum_{n \in \Omega(m)} r_{mn} \mathbf{v}_n \right)$$

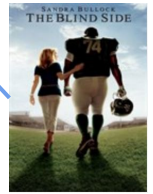
$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}_m} = \mathbf{0}$$

$\mathbf{u}_3$

r_{33}

$\mathbf{v}_3$

r_{35}



$\mathbf{v}_5$

$$\mathbf{u}_m = \left(\lambda \sum_{n \in \Omega(m)} \mathbf{v}_n \mathbf{v}_n^T + \tau \mathbf{I} \right)^{-1} \left(\lambda \sum_{n \in \Omega(m)} r_{mn} \mathbf{v}_n \right)$$

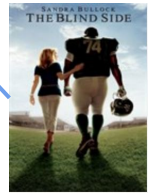
$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}_m} = \mathbf{0}$$

$\mathbf{u}_3$

r_{33}

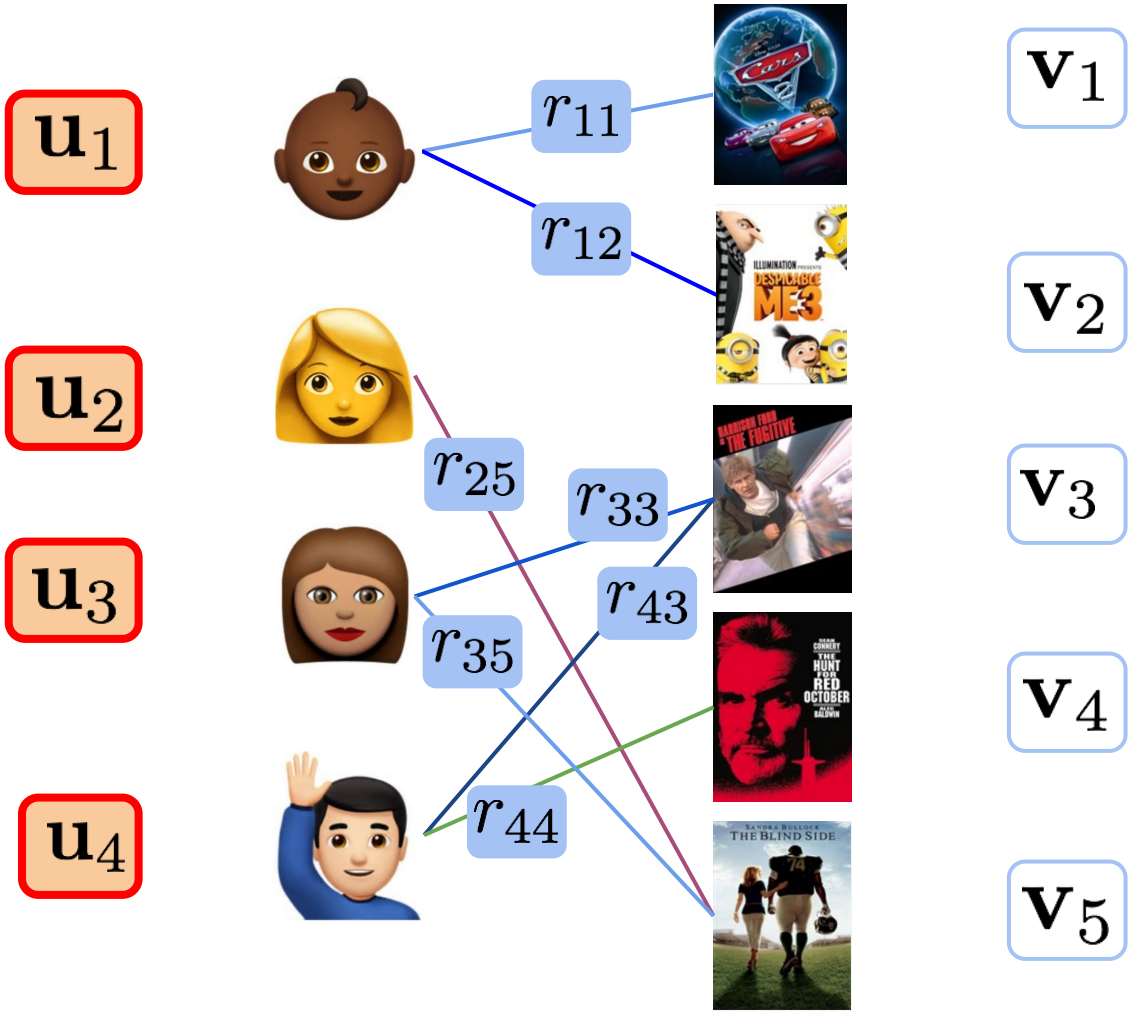
$\mathbf{v}_3$

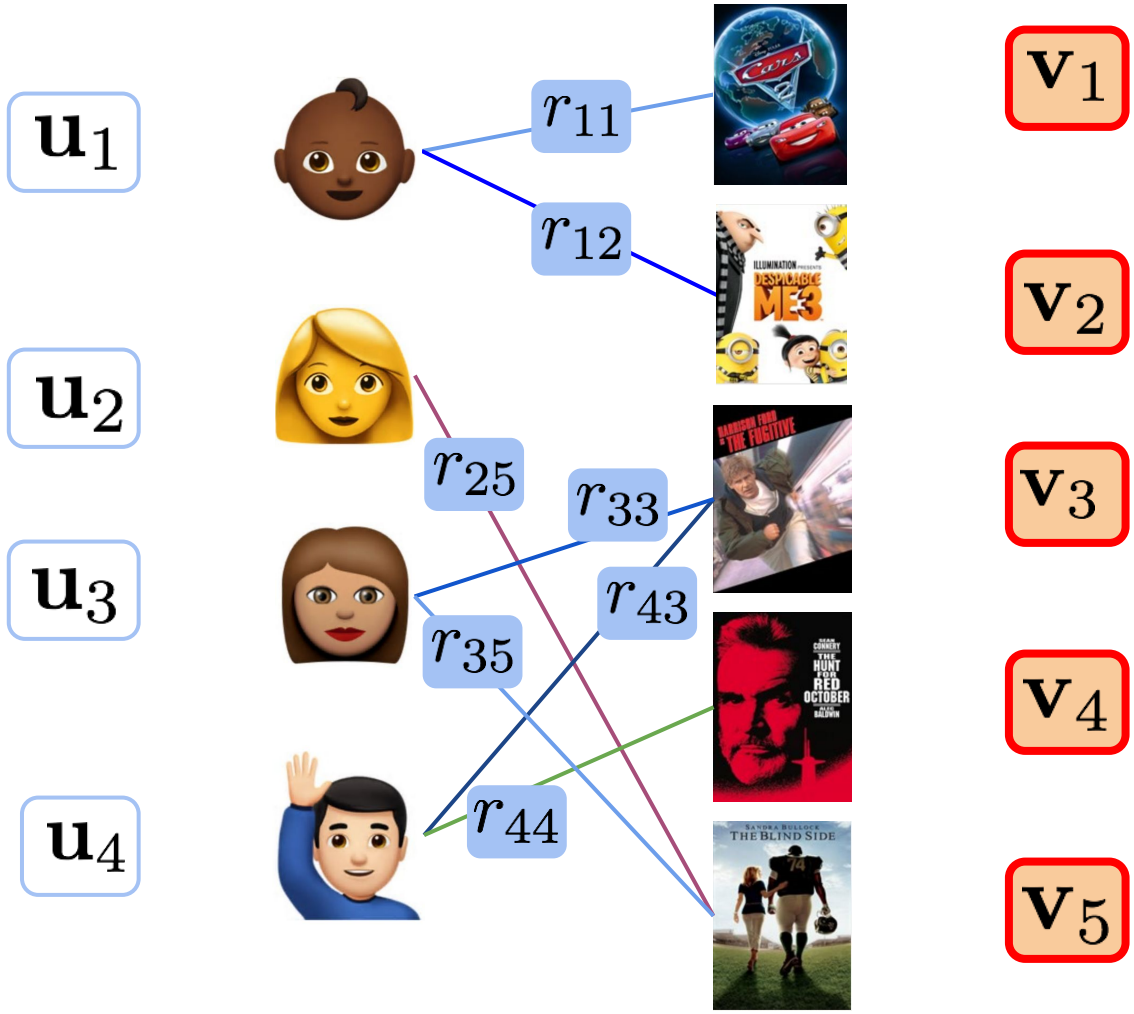
r_{35}

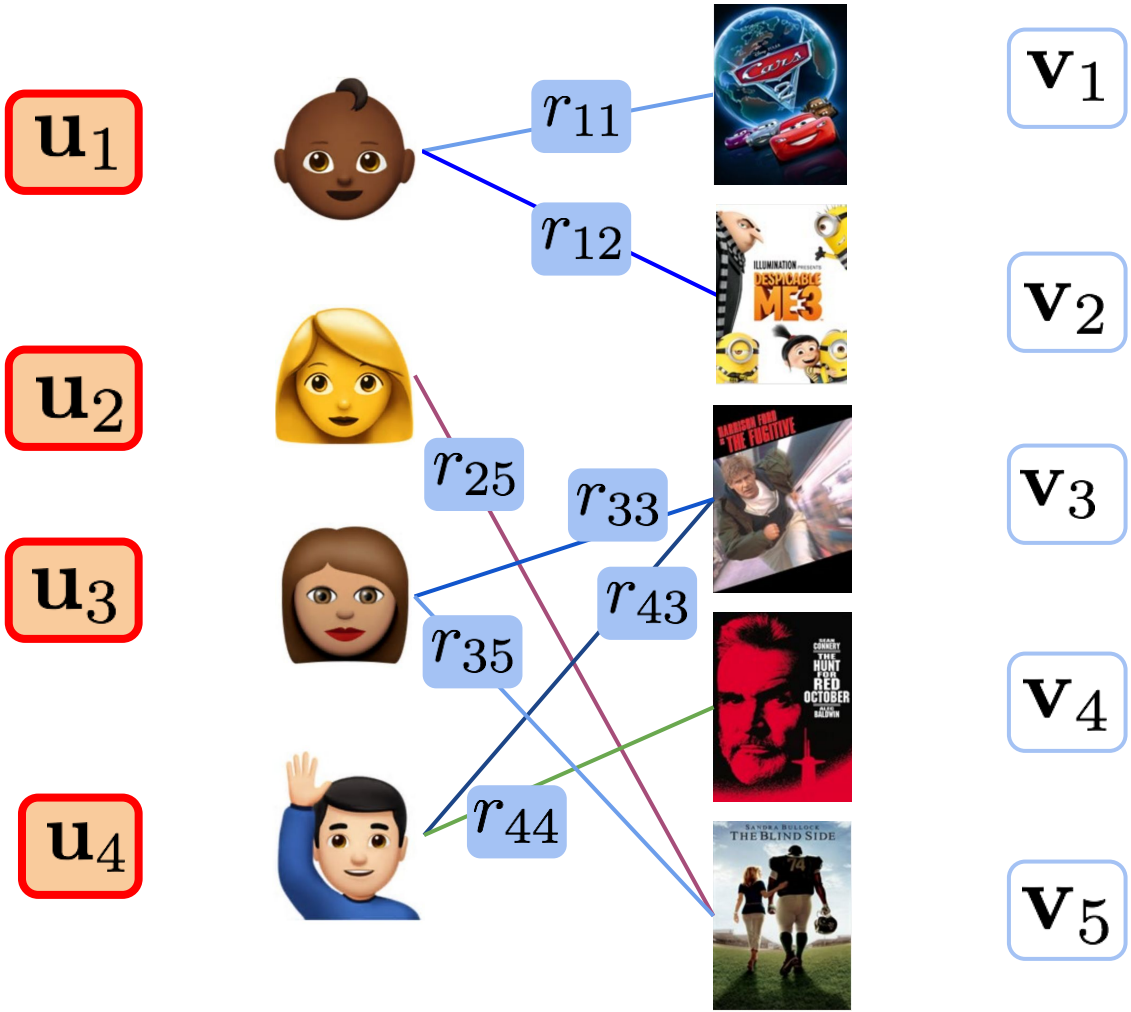


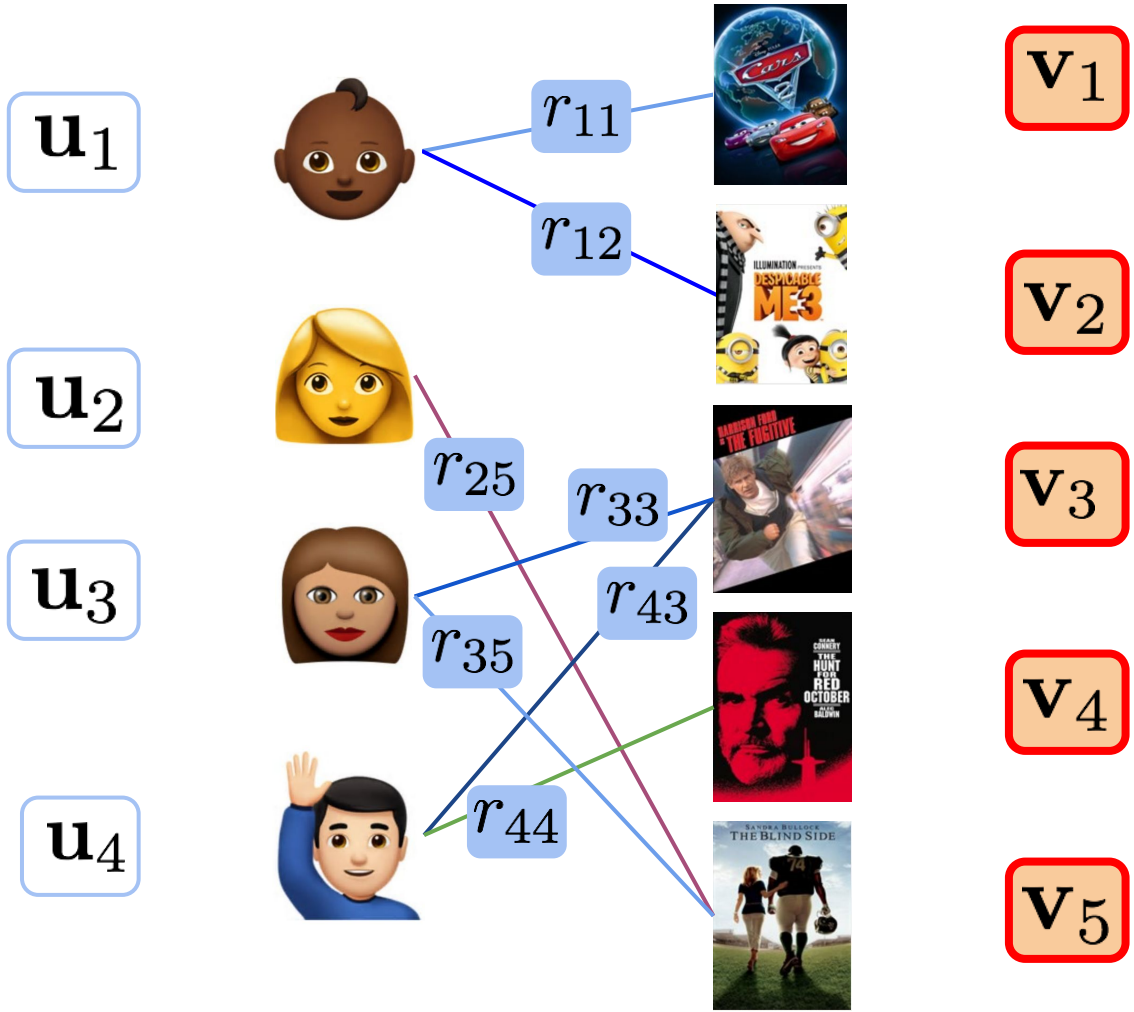
$\mathbf{v}_5$

We can set these partial gradients to zero, in **parallel**, for each of our millions of users...


$\mathbf{u}_1$
$\mathbf{u}_2$
$\mathbf{u}_3$
$\mathbf{u}_4$
r_{11}
r_{12}
r_{25}
r_{33}
r_{43}
r_{35}
r_{44}
$\mathbf{v}_1$
$\mathbf{v}_2$
$\mathbf{v}_3$
$\mathbf{v}_4$
$\mathbf{v}_5$
Cars 2
ILLUMINATION presents
DESPICABLE ME 3
HARRISON FORD
THE FUGITIVE
SEAN CONNERY
THE HUNT FOR RED OCTOBER
ALEC BALDWIN
SANDRA BULLOCK
THE BLIND SIDE


$\mathbf{u}_1$
$\mathbf{u}_2$
$\mathbf{u}_3$
$\mathbf{u}_4$
r_{11}
r_{12}
r_{25}
r_{33}
r_{43}
r_{35}
r_{44}
$\mathbf{v}_1$
$\mathbf{v}_2$
$\mathbf{v}_3$
$\mathbf{v}_4$
$\mathbf{v}_5$
Cars 2
ILLUMINATION presents
DESPICABLE ME 3
HARRISON FORD
THE FUGITIVE
SEAN CONNERY
THE HUNT FOR RED OCTOBER
ALEC BALDWIN
SANDRA BULLOCK
THE BLIND SIDE


$\mathbf{u}_1$
$\mathbf{u}_2$
$\mathbf{u}_3$
$\mathbf{u}_4$
r_{11}
r_{12}
r_{25}
r_{33}
r_{43}
r_{35}
r_{44}
$\mathbf{v}_1$
$\mathbf{v}_2$
$\mathbf{v}_3$
$\mathbf{v}_4$
$\mathbf{v}_5$
Cars 2
ILLUMINATION presents
DESPICABLE ME 3
HARRISON FORD
THE FUGITIVE
SEAN CONNERY
THE HUNT FOR RED OCTOBER
ALEC BALDWIN
SANDRA BULLOCK
THE BLIND SIDE


$\mathbf{u}_1$
$\mathbf{u}_2$
$\mathbf{u}_3$
$\mathbf{u}_4$
r_{11}
r_{12}
r_{25}
r_{33}
r_{43}
r_{35}
r_{44}
$\mathbf{v}_1$
$\mathbf{v}_2$
$\mathbf{v}_3$
$\mathbf{v}_4$
$\mathbf{v}_5$
Cars 2
ILLUMINATION presents
DESPICABLE ME 3
HARRISON FORD
THE FUGITIVE
SEAN CONNERY
THE HUNT FOR RED OCTOBER
ALEC BALDWIN
SANDRA BULLOCK
THE BLIND SIDE

Basic structure

```
repeat:
    loop over users in parallel:
        update user bias
        update user trait vector u

    loop over items in parallel:
        update item bias
        update item trait vector v
```

Bipartite graphs and distributed, parallel computations

Iterative least squares

- Embarrassingly parallel updates
- Distributed compute architectures (spread over many nodes in a data centre)

This gives us an idea of what physical architectures can support these *kinds* of algorithms

Class exercise

What does the update of $b_m^{(u)}$ look like? I.e. if we had:

$$\mathcal{L} = -\frac{\lambda}{2} \sum_{mn} \left(r_{mn} - (\mathbf{u}_m^T \mathbf{v}_n + b_m^{(u)} + b_n^{(i)}) \right)^2 - \frac{\gamma}{2} b_m^{(u)\,2} + \cdots$$

sum over *m* here 😊 will fix

Class exercise

What does the update of $b_m^{(u)}$ look like? I.e. if we had:

$$\mathcal{L} = -\frac{\lambda}{2}\sum_{mn}\left(r_{mn} - (\mathbf{u}_m^T\mathbf{v}_n + b_m^{(u)} + b_n^{(i)})\right)^2 - \frac{\gamma}{2}b_m^{(u)\,2} + \cdots$$

sum over *m* here 😊 will fix

$$b_m^{(u)} = \frac{\lambda\sum_{n\in\Omega(m)}\left(r_{mn} - (\mathbf{u}_m^T\mathbf{v}_n + b_n^{(i)})\right)}{\lambda\left|\Omega(m)\right| + \gamma}$$

Class exercise

What does the update of $\mathbf{u}_m$ look like with biases? I.e. if we had:

$$\mathcal{L} = -\frac{\lambda}{2}\sum_{mn}\left(r_{mn} - (\mathbf{u}_m^T\mathbf{v}_n + b_m^{(u)} + b_n^{(i)})\right)^2 - \frac{\tau}{2}\sum_m \mathbf{u}_m^T\mathbf{u}_m + \cdots$$

Class exercise

What does the update of $\mathbf{u}_m$ look like with biases? I.e. if we had:

$$\mathcal{L} = -\frac{\lambda}{2}\sum_{mn}\left(r_{mn} - (\mathbf{u}_m^T\mathbf{v}_n + b_m^{(u)} + b_n^{(i)})\right)^2 - \frac{\tau}{2}\sum_m \mathbf{u}_m^T\mathbf{u}_m + \cdots$$

$$\mathbf{u}_m = \left(\lambda \sum_{n\in\Omega(m)} \mathbf{v}_n\mathbf{v}_n^T + \tau\mathbf{I}\right)^{-1}\left(\lambda \sum_{n\in\Omega(m)} \mathbf{v}_n(r_{mn} - b_m^{(u)} - b_n^{(i)})\right)$$

what's the idea here?

Embeddings

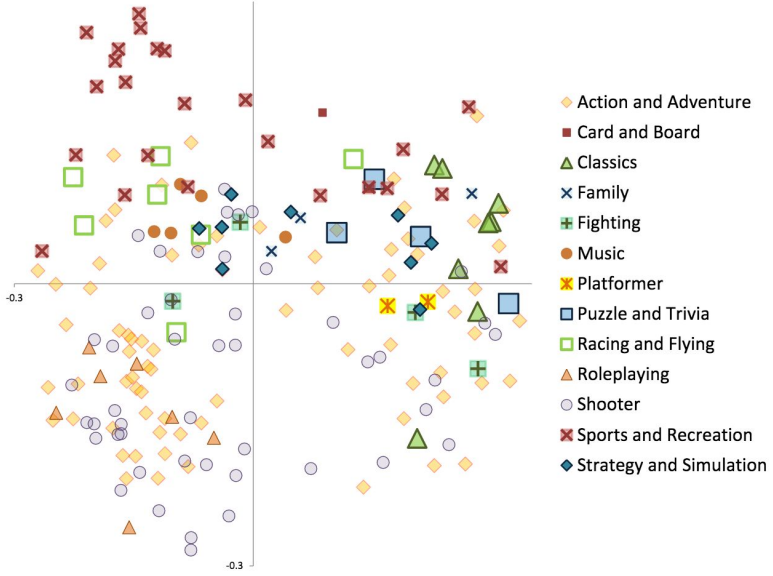
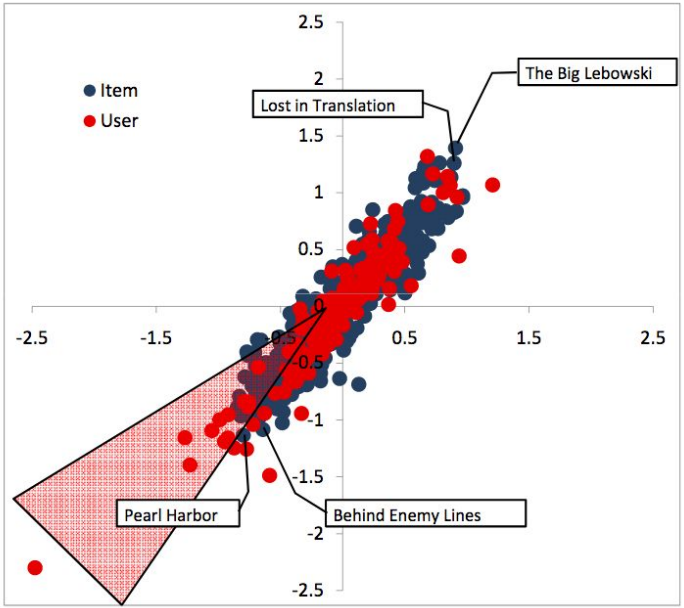
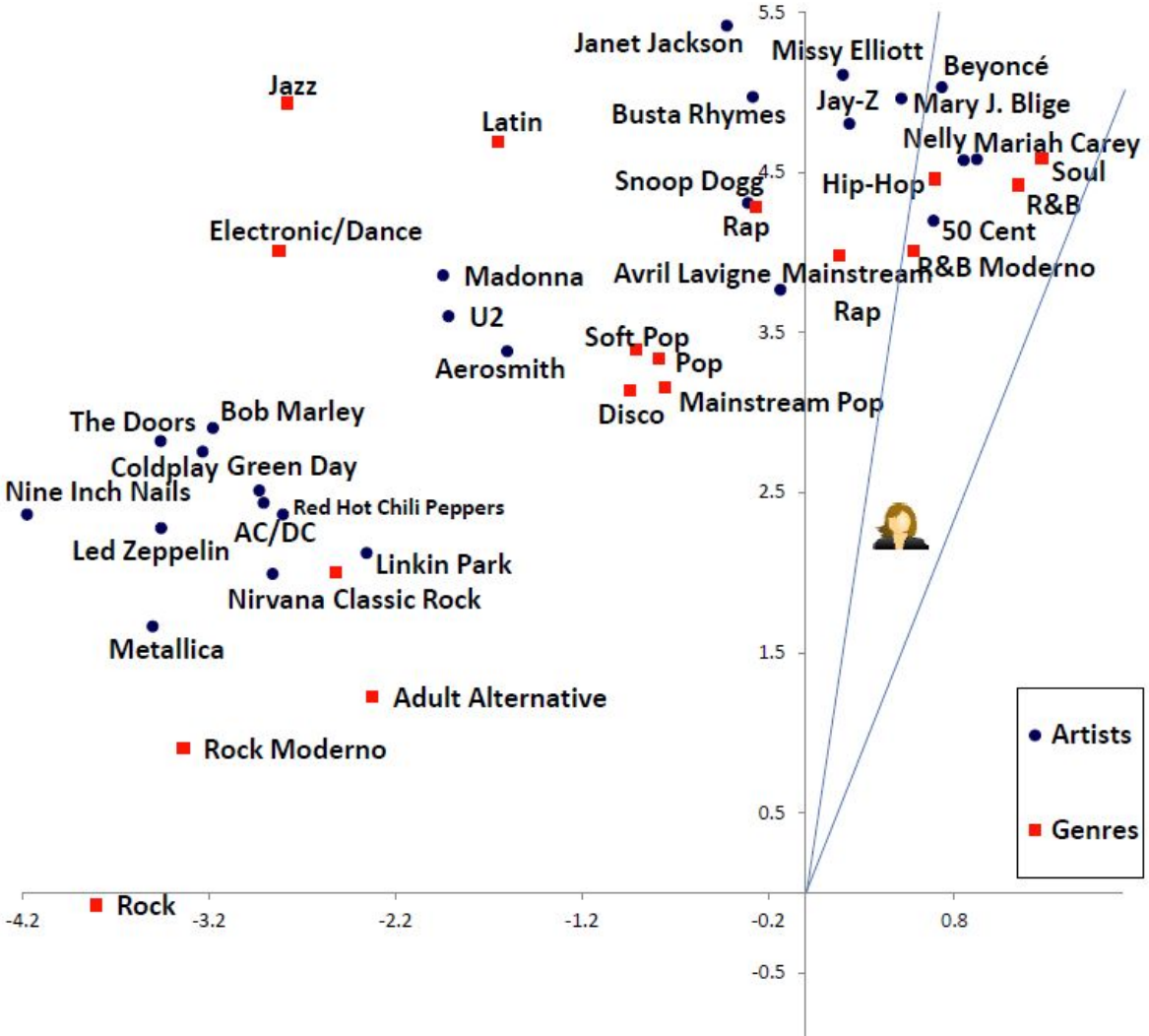


Figure 2: Game feature vectors embedded in $\mathbb{R}^2$, tagged by genre.

The Xbox Recommender System, Koenigstein et al, RecSys 2012



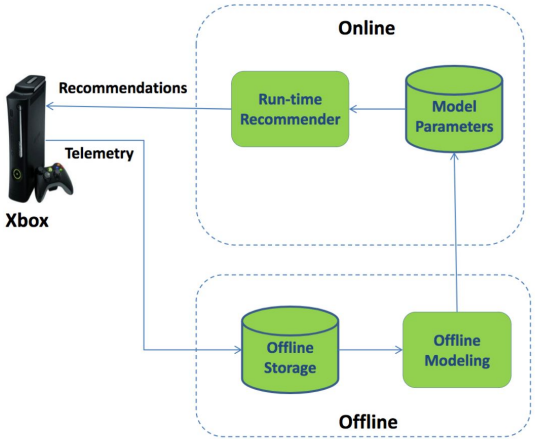
Matchbox: Large Scale Bayesian Recommendations, Stern et al, WWW 2009


Janet Jackson
Missy Elliott
Beyoncé
Jazz
Jay-Z
Mary J. Blige
Busta Rhymes
Latin
Nelly
Mariah Carey
Soul
Snoop Dogg
Hip-Hop
R&B
Rap
Electronic/Dance
50 Cent
Madonna
Avril Lavigne
Mainstream
R&B Moderno
Rap
U2
Soft Pop
Pop
Aerosmith
Mainstream Pop
Disco
The Doors
Bob Marley
Coldplay
Green Day
Nine Inch Nails
Red Hot Chili Peppers
AC/DC
Led Zeppelin
Linkin Park
Nirvana
Classic Rock
Metallica
Adult Alternative
Rock Moderno
Rock
Artists
Genres
5.5
4.5
3.5
2.5
1.5
0.5
-0.5
-4.2
-3.2
-2.2
-1.2
-0.2
0.8

Rotational invariance of the embeddings

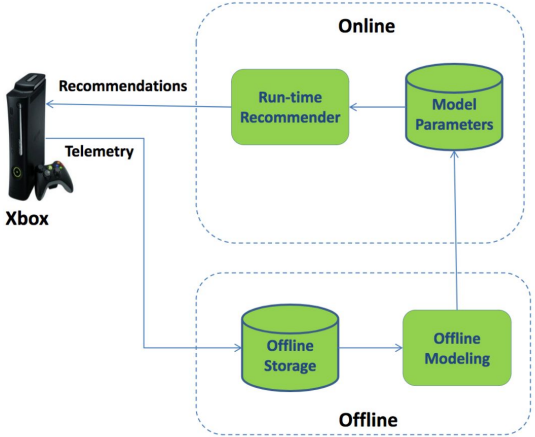
Some good properties

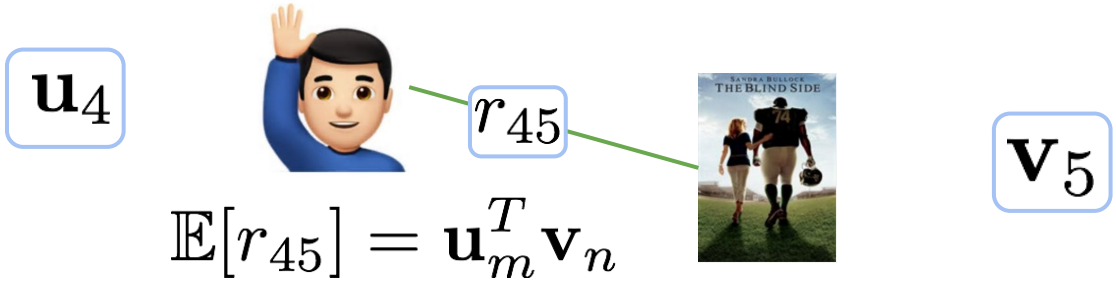
- In the *simplest* version, we only need to
  - store one vector for each user
  - store one vector for each item
  - load a user's vector at runtime
  - compute predictions over all items
  - ...and return something (like the top)



Making predictions

- Store vectors for each user and item
- At runtime, compute inner products and rank larges





Evaluation: train, validation and test sets

Train, Validation and Test Datasets

- Train Dataset – The sample of data used to fit the model. This is the actual data we use to train the model and learn parameters
- Validation Dataset – The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters.

  Models often have many hyperparameters (e.g., dimensionality, number of layers, architecture choices, optimization choices, etc.). The validation set is used only after the model is trained on the training dataset. It is used to evaluate the quality of the model under different hyper-parameter choices. Often the test dataset is used for tuning hyperparameters. However, as the model becomes biased towards this dataset, it is impossible to determine the actual quality of the model without a separate set.
- Test Datasets – The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.

  The test set is used once at the very end to evaluate the actual performance of a recommender systems.

Train, Validation and Datasets Test Cont.

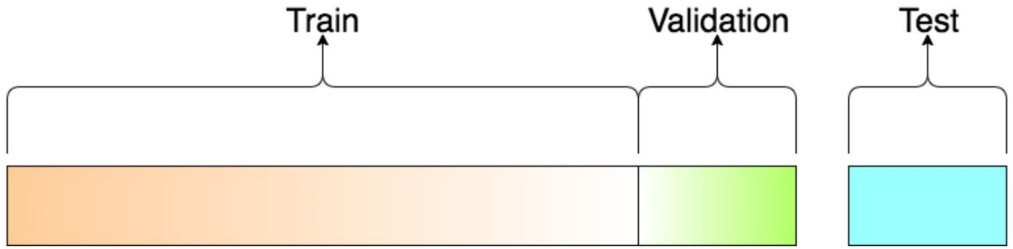
Most machine learning datasets are created by randomly splitting the data into train, validation and test.

Cross validation can be achieved by repeating the this split multiple times.

Recommendations datasets are time-dependent

- The split usually follows the actual timeline of events
- Random split usually yield an “easier” datasets that are not “realistic”

Towards Data Science
Sharing concepts, ideas, and codes.



A visualisation of the splits

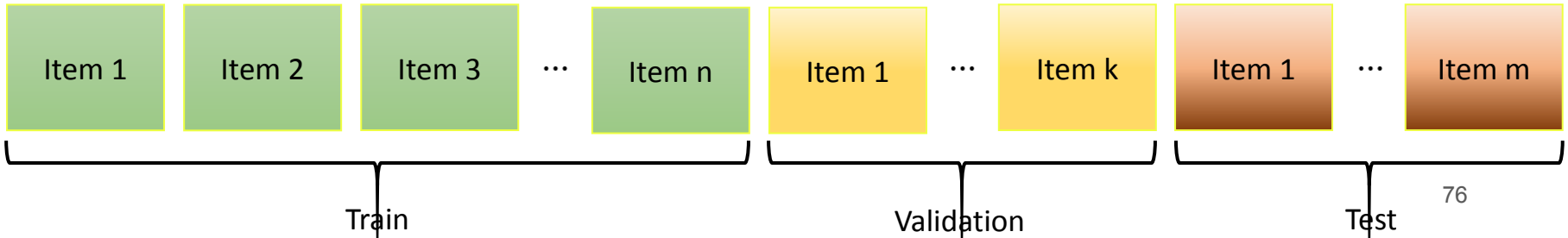
Dataset Partition - Coping with Long Tail Distributions

Recommendation datasets often exhibit a power-law distribution of ratings per items and per users.

We do not want the recommender system to be biased towards “heavy” users. Instead, we want the recommender system to “care” about all the users the same.

Removing ratings from heavy users or items is equivalent to deleting potentially useful data and changing user-item relations.

In many cases (e.g., Netflix, KDD Cup 2011), the last *m* ratings of each users are left for the validation set and the last *n* ratings are left for the test set.



Practical 2

A model with biases only

Practical 2

✅ Download a **smaller** MovieLens dataset to develop and debug with, and again index it

✅ Split the small data set into a training and a test set.

You would need two sparse matrices with the same number of rows and columns:

- `data_by_user_train` and `data_by_user_test`
- `data_by_movie_train` and `data_by_movie_test`

Practical 2

✅ Build a model that finds maximum likelihood estimate for user + item biases with **alternating least squares**

✅ Plot the loss function (negative log likelihood) over training iterations, using `data_by_user_train`. It should go down monotonically!

Note that if you plotted the loss function using `data_by_movie_train`, the answers would be the same

✅ Plot the root mean squared error over training iterations, using `data_by_user_train`. What does it converge to?

✅ Plot the root mean squared error for `data_by_user_test` after each training iteration. What does it converge to? Does it follow the training error?

Practical 2: biases only

```
user_biases = np.zeros((M))
item_biases = np.zeros((N))

repeat:
    loop over users ~~in parallel~~:
        update user bias

    loop over items ~~in parallel~~:
        update item bias
```

Practical 2: biases only

✅ Your user + item biases code with **alternating least squares** will look like this:

```
user_biases = np.zeros((M))
item_biases = np.zeros((N))

repeat:
  for m in 0 .. M-1:
     bias = 0
     item_counter = 0
     for (n, r) in get_items_and_ratings_for_user(m):
       bias += lambda * (r - item_bias[n])
       item_counter += 1
    bias = bias / (lambda * item_counter + gamma)
    user_bias[m] = bias

  # now do the same for the items
  for n in 0 .. N-1:
    ...
```

After one (or a few) set(s) of updates, compute the current **loss** here on the **training** and **test** (or validation) set

…compute the current **RMSE** here on the **training** and **test** (or validation) set

Initializing U and V

Practical 3

A model with embeddings

Practical 3

✅ Add user and item factors (latent trait vectors or matrices **U** and **V**)

✅ Add a sensible regularizer to your loss function

✅ Build a model that finds maximum likelihood estimate for user + item biases + latent trait vectors / factors with **alternating least squares** using `data_by_user_train` and `data_by_movie_train`

✅ Plot the loss function (negative log likelihood) over training iterations using `data_by_user_train`. It should go down monotonically!

✅ Plot the root mean squared error over training iterations using `data_by_user_train` and `data_by_user_test` . What does it converge to?

Practical 3: with user and item vectors

✅ Your code with **alternating least squares** will look something like this:

```
repeat:
  for m in 0 .. M-1:
    ...
    for (n, r) in get_items_and_ratings_for_user(m):
      sum things to update bias
    user_bias[m] = ...

    for (n, r) in get_items_and_ratings_for_user(m):
      sum things to trait vector
    user_vector[m,:] = ...

  # now do the same for the items
  for n in 0 .. N-1:
    ...
```

First update the user bias **and then** update the user vector.

A common coding bug is where everything is put in one loop!

Practical 4

Going big, training and testing

Practical 4

✅ Split the **MovieLens 25M (or 32M) dataset into a training and a test set**.

We won't do cross-validation in this practical, but we will check whether we are overfitting on a test set! How will you create a sensible split?

✅ Adapt your code so that you only use the training set for optimizing parameters. Measure the RMSE on the training set and test set!

Plots the train and test RMSE on the same axis as you progress. Can you show where overfitting happens?

Is there more overfitting with K=20 latent dimensions than K=10? Or not? For which users? Is there underfitting anywhere? Do changes of λ and τ affect this?

✅ If you have settings of K, λ and τ that you like, train on the full dataset (next)

Practical 4

✅ **Train on all 25M (or 32M) ratings.** You may have to make your algorithm parallel to get this to train faster :)

✅ Show that if a user liked movie X (your choice) the recommendations Y make sense.

That is, create a “dummy user” that gave some movie like a “Lord of the Rings” five stars. Are the top recommendations other “Lord of the Rings” movies? You’ll need the 25M dataset for this to start looking really nice.

✅ Can you find and show which movies as polalizing?

Some polarizing that quickly narrows down a user’s taste once you know whether they like / dislike it. Typically movies with unusually long trait vectors **v**

Practical 4

Tricks of the trade:

Create a “dummy user” that gave some movie like a “Lord of the Rings” five stars. Are the top recommendations other “Lord of the Rings” movies? You’ll need the 25M dataset for this to start looking really nice. You used something like:

```
score_for_item[n] = user_trait_vector[m].inner(item_trait_vector[n]) + item_bias[n]
```

for all movies n. (Your recommendation score for a user is independent of the user bias, hence you didn’t add that.) **This will not look nice because some “high bias” or popular movies will invariably always sit at the top**. To increase the personalization component, try ranking with

```
score_for_item[n] = user_trait_vector[m].inner(item_trait_vector[n]) + 0.05 *
item_bias[n]
```

where the contribution of the item bias is downplayed.

✅ Experiment! What works?

Practical 4

Tricks of the trade:

After ranking on:

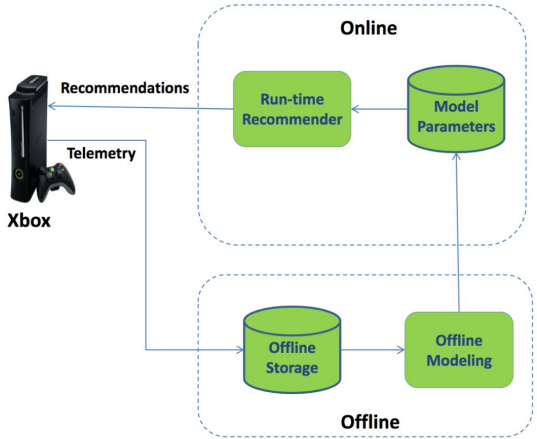
```
score_for_item[n] = user_trait_vector[m].inner(item_trait_vector[n])
+ 0.05 * item_bias[n]
```

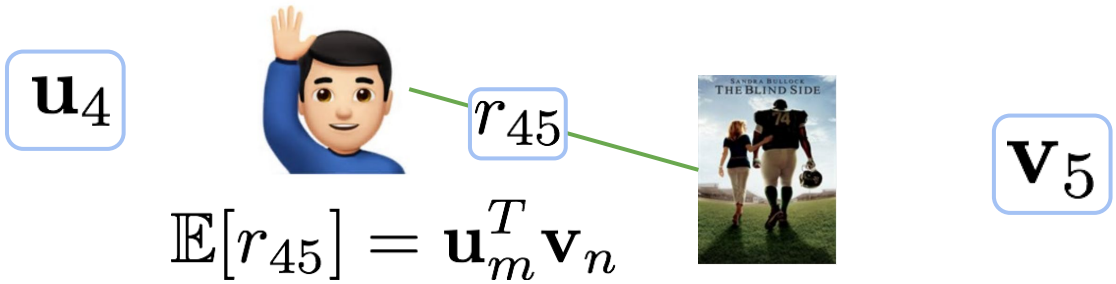
you may still see some strange movies at the top of the list. They would be movies with, say, *less than* 100 ratings in your training data set. Filter them out to get cleaner results.

✅ Experiment! Does it make a difference? Let's explore why on the next slides…

Making predictions

- Store vectors for each user and item
- At runtime, compute inner products and rank larges

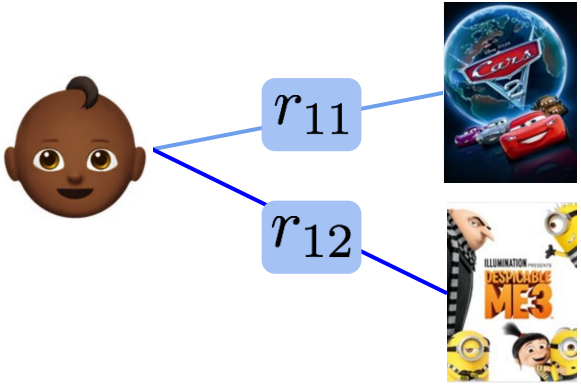


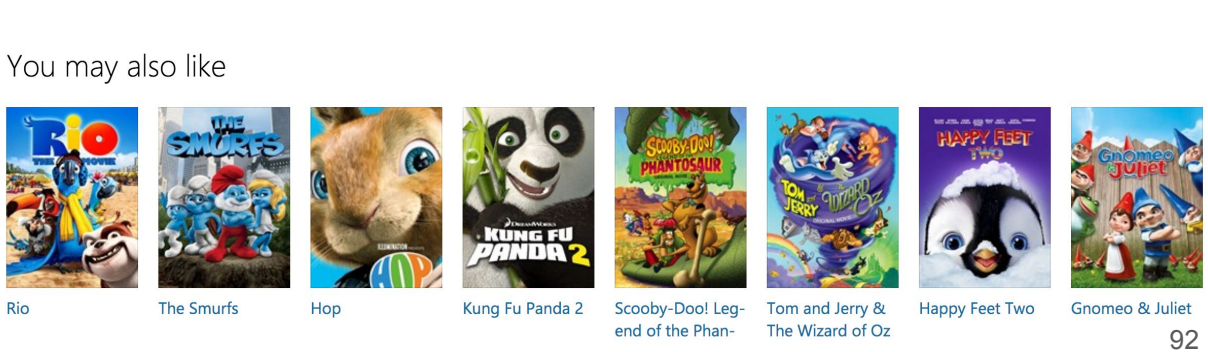


Making predictions

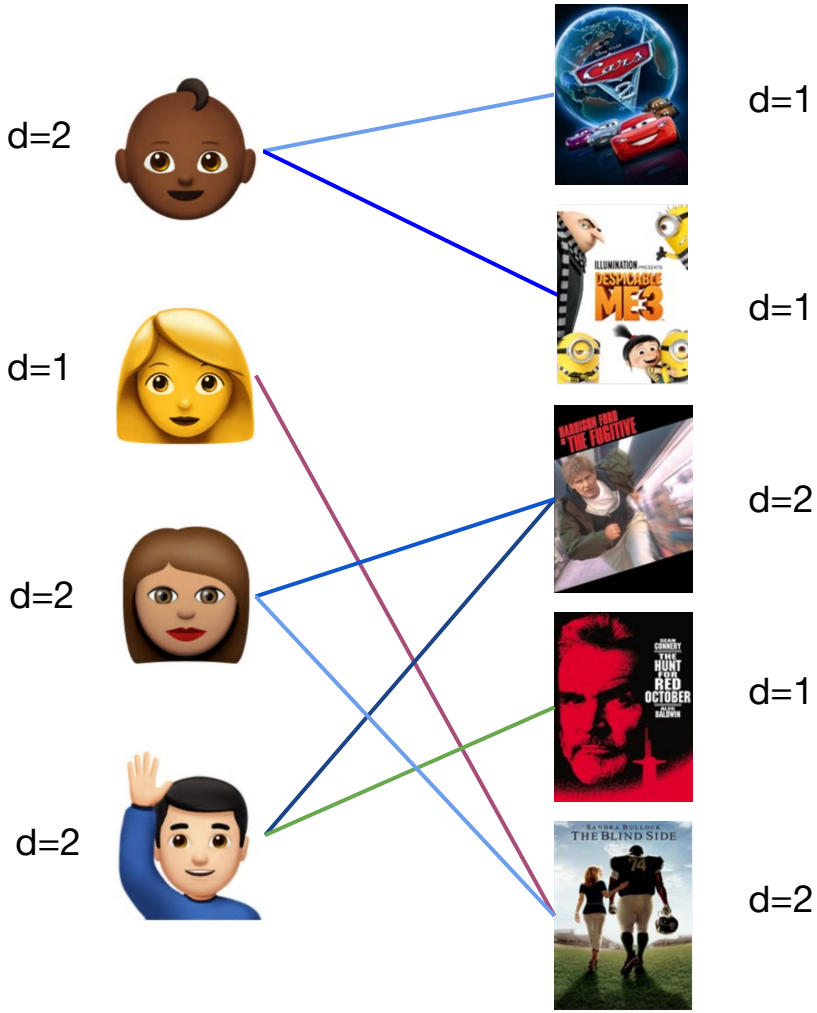
- ...but it will not look like this!

why not?

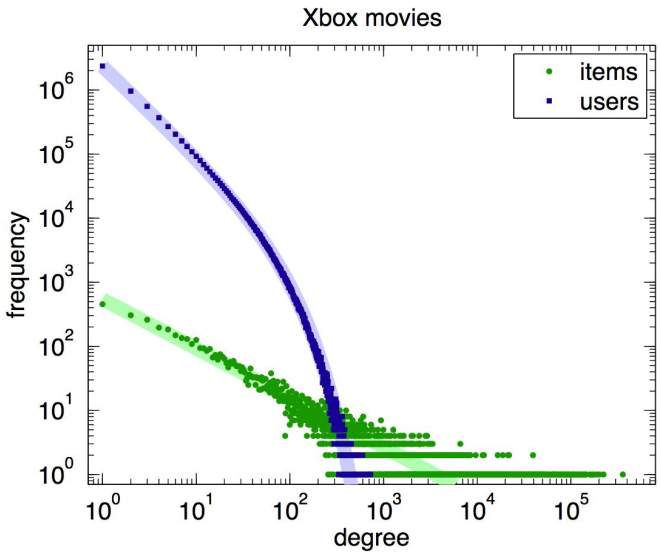


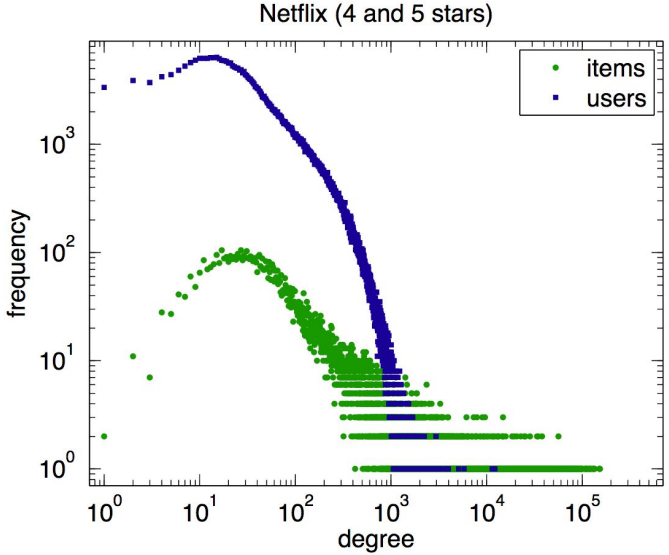


Degree distributions



Power laws


Xbox movies
items
users
frequency
degree
10^0
10^1
10^2
10^3
10^4
10^5
10^6


Netflix (4 and 5 stars)
items
users
frequency
degree
10^0
10^1
10^2
10^3
10^4
10^5

Making predictions

- ...but it will not look like this!

  why not?

- **Overfitting, somewhere!**
- Ranking, say, 100,000 movies… Some “rare” movies trickle to the top!
- Latent dimensionality for factors, e.g. $\mathbf{u}_m \in \mathbb{R}^{40}$

$$\mathbf{u}_m = \left(\lambda \sum_{n \in \Omega(m)} \mathbf{v}_n \mathbf{v}_n^T + \tau \mathbf{I} \right)^{-1} \left(\lambda \sum_{n \in \Omega(m)} r_{mn} \mathbf{v}_n \right)$$

could be small

$$p(\mathbf{R}, \mathbf{U}, \mathbf{V}, \mathbf{b}^{(u)}, \mathbf{b}^{(i)}) = \ldots$$

Averaging over the posterior distribution

Remedy: average over all plausible settings of **U** and **V**...

Marginalization!

$$
\begin{aligned}
p(r_{45}|\mathbf{R}) &= \int p(r_{45}, \mathbf{u}_4, \mathbf{v}_5|\mathbf{R})\, \mathrm{d}\mathbf{u}_4 \mathrm{d}\mathbf{v}_5 \\
&= \int p(r_{45}|\mathbf{u}_4, \mathbf{v}_5)\, p(\mathbf{u}_4, \mathbf{v}_5|\mathbf{R})\, \mathrm{d}\mathbf{u}_4 \mathrm{d}\mathbf{v}_5 \\
&= \int p(r_{45}|\mathbf{u}_4, \mathbf{v}_5)\, p(\mathbf{U}, \mathbf{V}|\mathbf{R})\, \mathrm{d}\mathbf{U} \mathrm{d}\mathbf{V}
\end{aligned}
$$

Averaging over the posterior distribution

Remedy: average over all plausible settings of **U** and **V**...

Bayes's theorem

$$p(\mathbf{U}, \mathbf{V}|\mathbf{R}) = \frac{p(\mathbf{R}|\mathbf{U}, \mathbf{V})p(\mathbf{U})p(\mathbf{V})}{p(\mathbf{R})}$$

Averaging over the posterior distribution

Remedy: average over all plausible settings of **U** and **V**...

Marginalization!

$$p(r_{45}|\mathbf{R}) = \int p(r_{45}, \mathbf{u}_4, \mathbf{v}_5|\mathbf{R})\,\mathrm{d}\mathbf{u}_4\mathrm{d}\mathbf{v}_5$$

$$= \int p(r_{45}|\mathbf{u}_4, \mathbf{v}_5)\, p(\mathbf{u}_4, \mathbf{v}_5|\mathbf{R})\,\mathrm{d}\mathbf{u}_4\mathrm{d}\mathbf{v}_5$$

$$= \int p(r_{45}|\mathbf{u}_4, \mathbf{v}_5)\, p(\mathbf{U}, \mathbf{V}|\mathbf{R})\,\mathrm{d}\mathbf{U}\mathrm{d}\mathbf{V}$$

With a "point mass" on **U** and **V**, like we had earlier, this is still analytically tractable. Now it is not.

r distribution

s of **U** and **V**...

- We can approximate this integral
- Draw samples from the **posterior** (Markov chain Monte Carlo; block Gibbs sampling), **offline**
- Store samples for each user and item on disk, and approximate the integral
- Problem: memory on disk scales with the number of samples
- Problem: **online** evaluation time scales with the number of samples

$$|\mathbf{R})\,\mathrm{d}\mathbf{u}_4\mathrm{d}\mathbf{v}_5$$

$$5)\,p(\mathbf{u}_4,\mathbf{v}_5|\mathbf{R})\,\mathrm{d}\mathbf{u}_4\mathrm{d}\mathbf{v}_5$$

$$= \int p(r_{45}|\mathbf{u}_4,\mathbf{v}_5)\,p(\mathbf{U},\mathbf{V}|\mathbf{R})\,\mathrm{d}\mathbf{U}\mathrm{d}\mathbf{V}$$

Mean field approximation

We want to approximate

$$p(\mathbf{U}, \mathbf{V}|\mathbf{R}) = \frac{p(\mathbf{R}|\mathbf{U}, \mathbf{V})p(\mathbf{U})p(\mathbf{V})}{p(\mathbf{R})}$$

with a factorized distribution, so that we now keep a mean and variance per dimension...

...so that we can still store a vector (or now, two vectors) of numbers per user and item (small memory footprint on disk)

Mean field variational inference

Mean field approximation

Optimize a variational lower bound to the log marginal likelihood:

$$\log p(\mathbf{R}) = \log \int p(\mathbf{R}|\mathbf{U}, \mathbf{V}) p(\mathbf{U}) p(\mathbf{V}) \, \mathrm{d}\mathbf{U} \mathrm{d}\mathbf{V}$$

Mean field approximation

Optimize a variational lower bound to the log marginal likelihood:

$$\log p(\mathbf{R}) = \log \int p(\mathbf{R}|\mathbf{U},\mathbf{V})p(\mathbf{U})p(\mathbf{V})\,\mathrm{d}\mathbf{U}\mathrm{d}\mathbf{V}$$

$$= \log \int q(\mathbf{U})q(\mathbf{V})\frac{p(\mathbf{R}|\mathbf{U},\mathbf{V})p(\mathbf{U})p(\mathbf{V})}{q(\mathbf{U})q(\mathbf{V})}\,\mathrm{d}\mathbf{U}\mathrm{d}\mathbf{V}$$

Mean field approximation

Optimize a variational lower bound to the log marginal likelihood:

$$\begin{aligned}
\log p(\mathbf{R}) &= \log \int p(\mathbf{R}|\mathbf{U},\mathbf{V}) p(\mathbf{U}) p(\mathbf{V}) \, \mathrm{d}\mathbf{U}\mathrm{d}\mathbf{V} \\
&= \log \int q(\mathbf{U}) q(\mathbf{V}) \frac{p(\mathbf{R}|\mathbf{U},\mathbf{V}) p(\mathbf{U}) p(\mathbf{V})}{q(\mathbf{U}) q(\mathbf{V})} \, \mathrm{d}\mathbf{U}\mathrm{d}\mathbf{V} \\
&\geq \int q(\mathbf{U}) q(\mathbf{V}) \log \frac{p(\mathbf{R}|\mathbf{U},\mathbf{V}) p(\mathbf{U}) p(\mathbf{V})}{q(\mathbf{U}) q(\mathbf{V})} \, \mathrm{d}\mathbf{U}\mathrm{d}\mathbf{V}
\end{aligned}$$

Maximization is over $q(\mathbf{U})$ and $q(\mathbf{V})$

$$q(\mathbf{u}_1) = \prod_{k=1}^{K} \mathcal{N}(u_{1k}; \mu_{1k}, \sigma_{1k}^2)$$

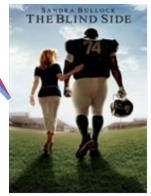
$$q(\mathbf{u}_2) = \prod_{k=1}^{K} \mathcal{N}(u_{2k}; \mu_{2k}, \sigma_{2k}^2)$$

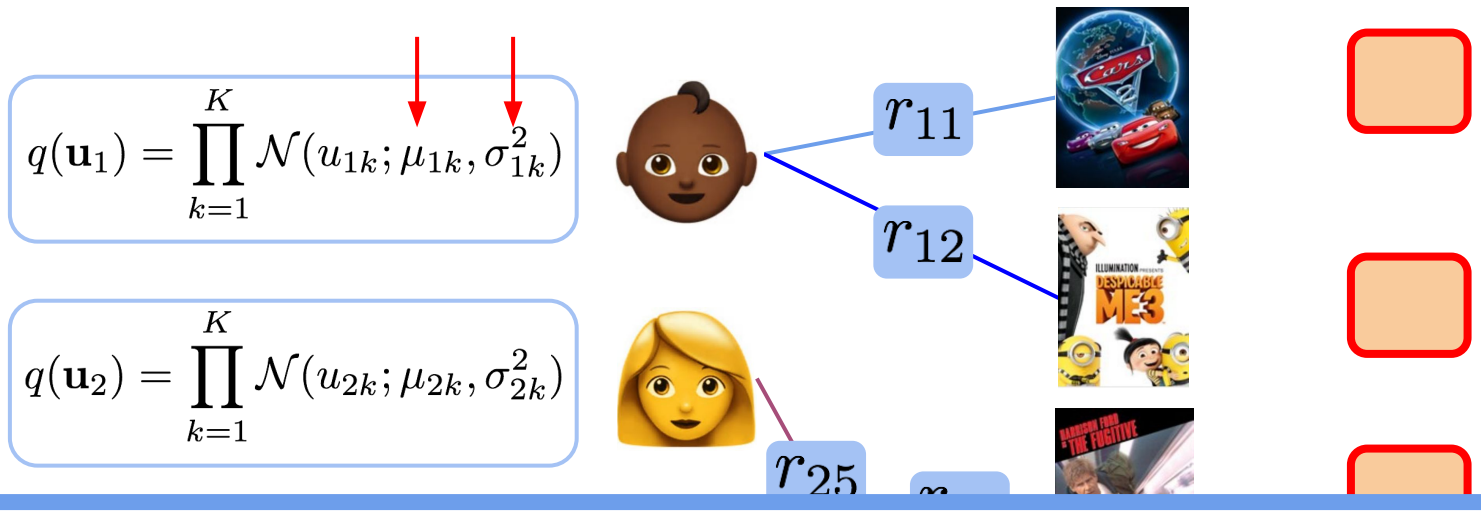



r_{11} r_{12} r_{25} r_{33} r_{43} r_{35} r_{44}









- **Conjugate exponential family**
- Component-wise gradient descent on distributions *q* have a particularly nice **closed form solution**

  (a bit like the maximum a posteriori case earlier)

  ...and has the same form / complexity as **Gibbs sampling** updates if we were to sample from the posterior

Making predictions

We capture some notion of posterior uncertainty

$$\int p(r_{45}|\mathbf{u}_4, \mathbf{v}_5)\, p(\mathbf{U}, \mathbf{V}|\mathbf{R})\, \mathrm{d}\mathbf{U}\mathrm{d}\mathbf{V} \approx \int p(r_{45}|\mathbf{u}_4, \mathbf{v}_5)\, q(\mathbf{U})\, q(\mathbf{V})\, \mathrm{d}\mathbf{U}\mathrm{d}\mathbf{V}$$

$$= \int p(r_{45}|\mathbf{u}_4, \mathbf{v}_5)\, q(\mathbf{u}_4)\, q(\mathbf{v}_5)\, \mathrm{d}\mathbf{u}_4\mathrm{d}\mathbf{v}_5$$

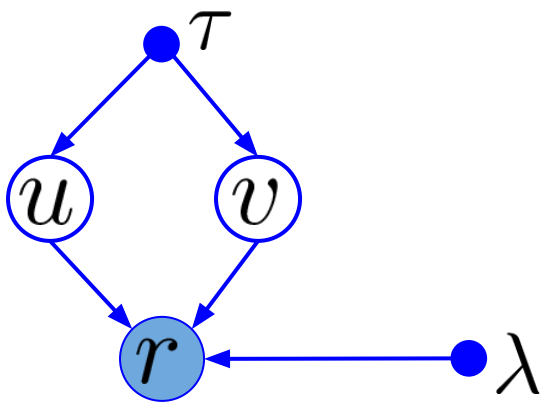
Fully factorized posterior approximation

Inference

A very simple example!

One user, one item, one rating

$$
\begin{aligned}
p(r, u, v) &= p(r|u, v)\, p(u)\, p(v) \\
&= \mathcal{N}(r; uv, \lambda^{-1})\, \mathcal{N}(u; 0, \tau^{-1})\, \mathcal{N}(v; 0, \tau^{-1})
\end{aligned}
$$

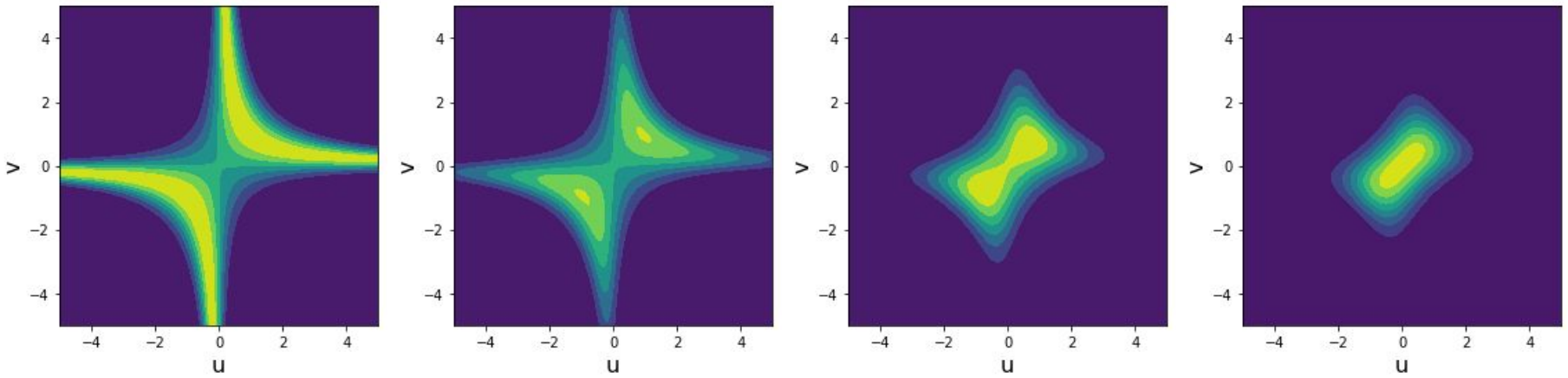


The joint density

$$p(r, u, v) = p(r|u, v)\, p(u)\, p(v)$$
$$= \mathcal{N}(r; uv, \lambda^{-1})\, \mathcal{N}(u; 0, \tau^{-1})\, \mathcal{N}(v; 0, \tau^{-1})$$

Increasing τ ...

$p(u, v|r)$

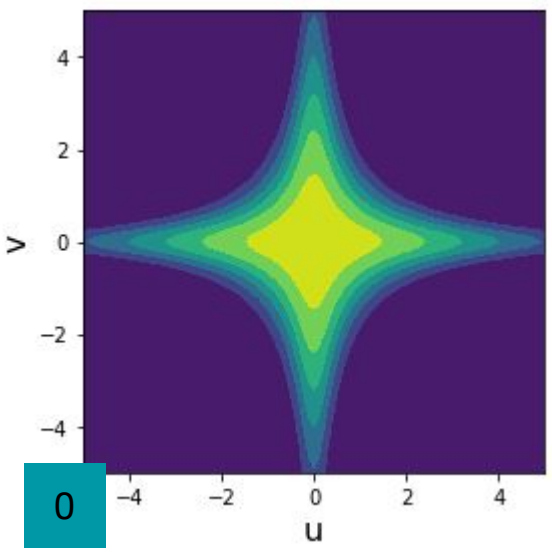


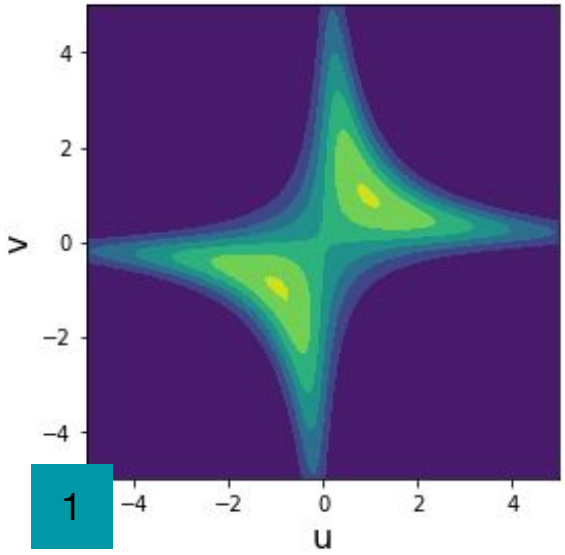
The joint density

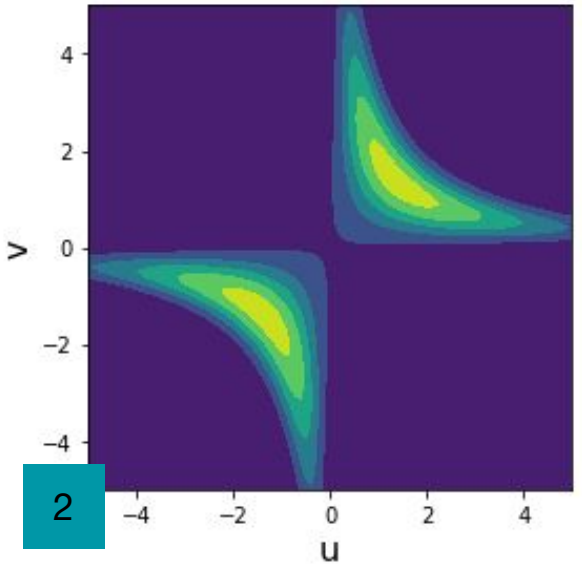
$$p(r, u, v) = p(r|u, v)\, p(u)\, p(v)$$
$$= \mathcal{N}(r; uv, \lambda^{-1})\, \mathcal{N}(u; 0, \tau^{-1})\, \mathcal{N}(v; 0, \tau^{-1})$$

Increasing r ...

$p(u, v|r)$







Class practical

Understanding the joint density

Recreate the figures :)

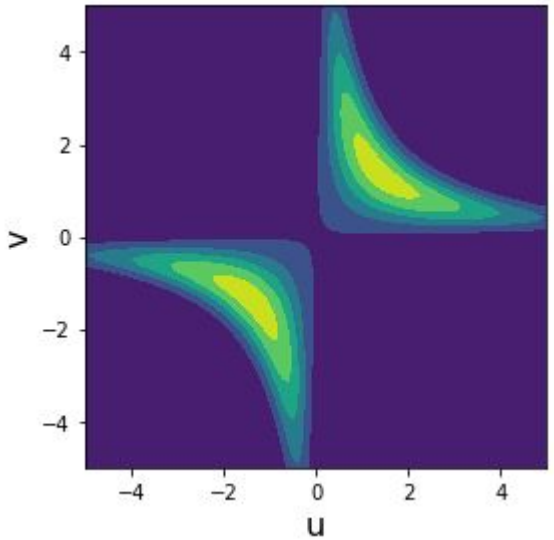
```
u = np.arange(-5, 5, 0.25)

v = np.arange(-5, 5, 0.25)

U, V = np.meshgrid(u, v)

P = np.exp(- 0.5 * tau * (U**2 + V**2) ... )

# correct and fill in ...!

surf = plt.contourf(U, V, P, ...)
```

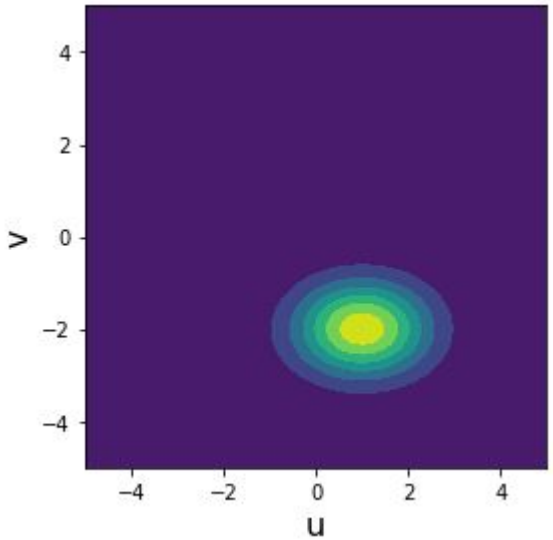
scratch space for lecture

Variational Bayes

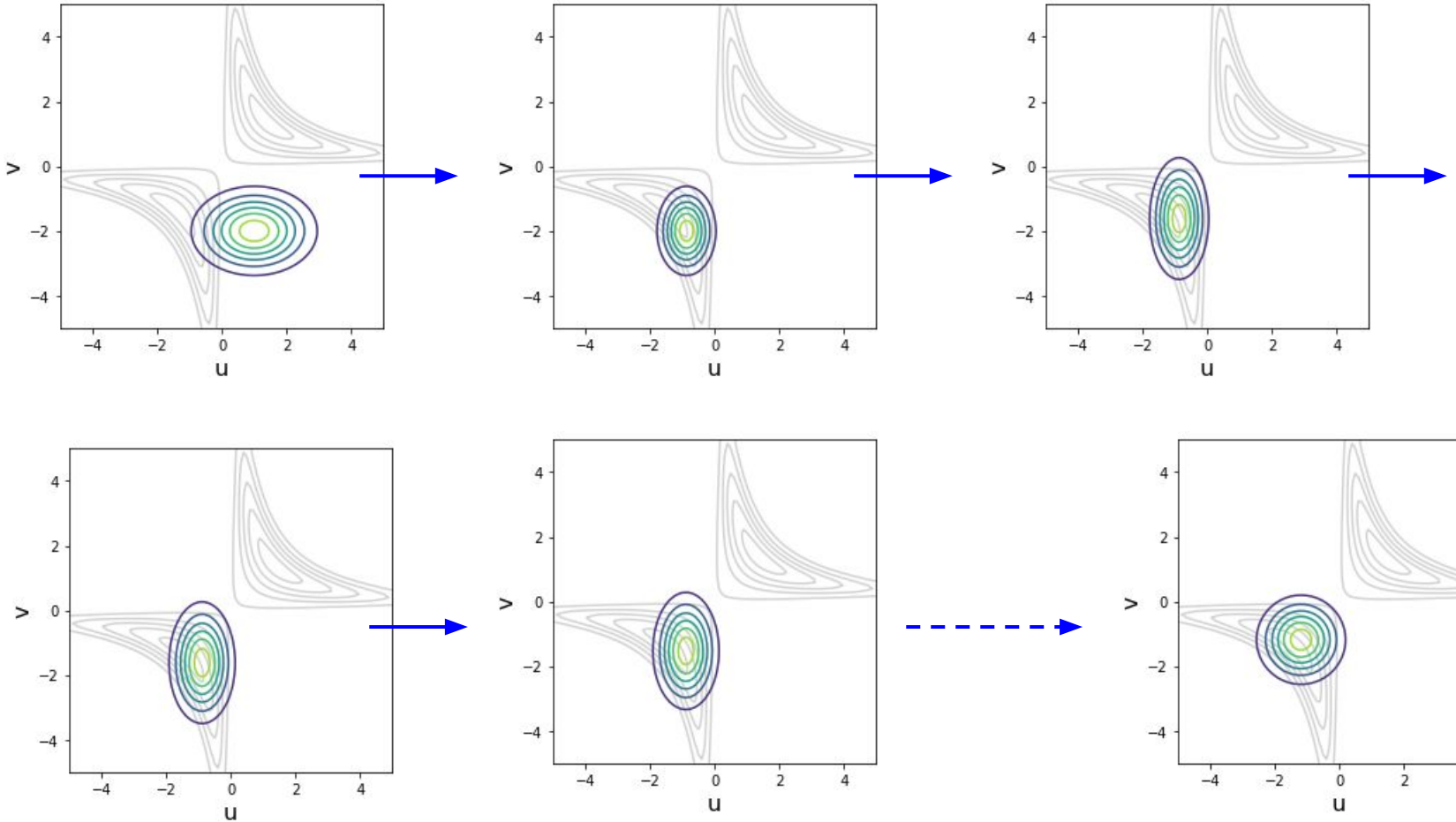
A variational approximation

$$p(u, v|r) \approx q(u)\, q(v)$$





$$q(u) = \mathcal{N}(u; \mu_u, \sigma_u^2)$$
$$q(v) = \mathcal{N}(v; \mu_v, \sigma_v^2)$$


v
u
4
2
0
−2
−4
−4
−2
0
2
4

Objective function

$$\min_{q(u)q(v)} D_{\mathrm{KL}}\Big(q(u)q(v) \,\Big\|\, p(u,v|r)\Big)$$

Objective function

$$\min_{q(u)q(v)} D_{\mathrm{KL}}\Big(q(u)q(v)\,\Big\|\,p(u,v|r)\Big)$$

$$\begin{aligned} D_{\mathrm{KL}}\Big(q(u)q(v)\,\Big\|\,p(u,v|r)\Big) &= \iint q(u)q(v)\log\frac{q(u)q(v)}{p(u,v|r)}\,\mathrm{d}u\,\mathrm{d}v \\ &= -\iint q(u)q(v)\log p(u,v|r)\,\mathrm{d}u\,\mathrm{d}v \\ &\quad + \int q(u)\log q(u)\,\mathrm{d}u + \int q(v)\log q(v)\,\mathrm{d}v \end{aligned}$$

Free-form minimization; let $q(u)$ be any non-negative function

$$\mathcal{F}[q(u)] = -\iint q(u)q(v)\log p(u,v|r)\,\mathrm{d}u\,\mathrm{d}v + \int q(u)\log q(u)\,\mathrm{d}u$$
$$+\lambda\left(\int q(u)\,\mathrm{d}u - 1\right)$$

Lagrange multiplier: q should integrate to one!

Free-form minimization; let $q(u)$ be any non-negative function

$$\mathcal{F}[q(u)] = -\iint q(u)q(v)\log p(u,v|r)\,\mathrm{d}u\,\mathrm{d}v + \int q(u)\log q(u)\,\mathrm{d}u$$
$$+\lambda\left(\int q(u)\,\mathrm{d}u - 1\right)$$

Lagrange multiplier: q should integrate to one!

$$\frac{\partial\mathcal{F}[q(u)]}{\partial q(u)} = -\int q(v)\log p(u,v|r)\,\mathrm{d}v + \log q(u) + 1 + \lambda$$

Functional derivative

Free-form minimization; let $q(u)$ be any non-negative function

$$\mathcal{F}[q(u)] = -\iint q(u)q(v)\log p(u,v|r)\,\mathrm{d}u\,\mathrm{d}v + \int q(u)\log q(u)\,\mathrm{d}u$$
$$+\lambda\left(\int q(u)\,\mathrm{d}u - 1\right)$$

Lagrange multiplier: q should integrate to one!

$$\frac{\partial\mathcal{F}[q(u)]}{\partial q(u)} = -\int q(v)\log p(u,v|r)\,\mathrm{d}v + \log q(u) + 1 + \lambda$$

Functional derivative

Set to zero…

$$\log q(u) = \mathbb{E}_{q(v)}\Big[\log p(u,v|r)\Big] - \lambda - 1$$

Free-form minimization; let $q(u)$ be any non-negative function

$$\mathcal{F}[q(u)] = -\iint q(u)q(v)\log p(u,v|r)\,\mathrm{d}u\,\mathrm{d}v + \int q(u)\log q(u)\,\mathrm{d}u + \lambda\left(\int q(u)\,\mathrm{d}u - 1\right)$$

Lagrange multiplier: q should integrate to one!

$$\frac{\partial\mathcal{F}[q(u)]}{\partial q(u)} = -\int q(v)\log p(u,v|r)\,\mathrm{d}v + \log q(u) + 1 + \lambda$$

Functional derivative

Set to zero…

$$\log q(u) = \mathbb{E}_{q(v)}\Big[\log p(u,v|r)\Big] - \lambda - 1$$

$$q(u) = \frac{\exp\left(\mathbb{E}_{q(v)}\Big[\log p(u,v|r)\Big]\right)}{\int \exp\left(\mathbb{E}_{q(v)}\Big[\log p(u,v|r)\Big]\right)\mathrm{d}u}$$

Normalization comes from solving for the Lagrange multiplier

(The Lagrange multiplier)

$$\log q(u) = \mathbb{E}_{q(v)}\Big[\log p(u, v|r)\Big] - \lambda - 1$$

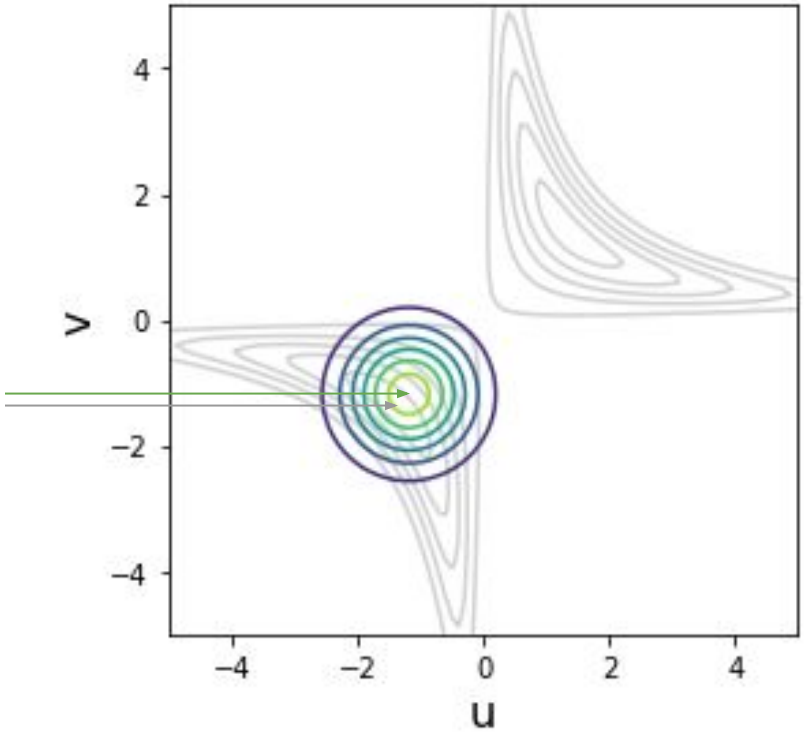
$$q(u) = \exp\Big(\mathbb{E}_{q(v)}\Big[\log p(u, v|r)\Big]\Big)\exp(-\lambda - 1)$$

$$\int q(u)\,\mathrm{d}u = \int \exp\Big(\mathbb{E}_{q(v)}\Big[\log p(u, v|r)\Big]\Big)\exp(-\lambda - 1)\,\mathrm{d}u$$

$$1 = \exp(-\lambda - 1)\int \exp\Big(\mathbb{E}_{q(v)}\Big[\log p(u, v|r)\Big]\Big)\,\mathrm{d}u$$

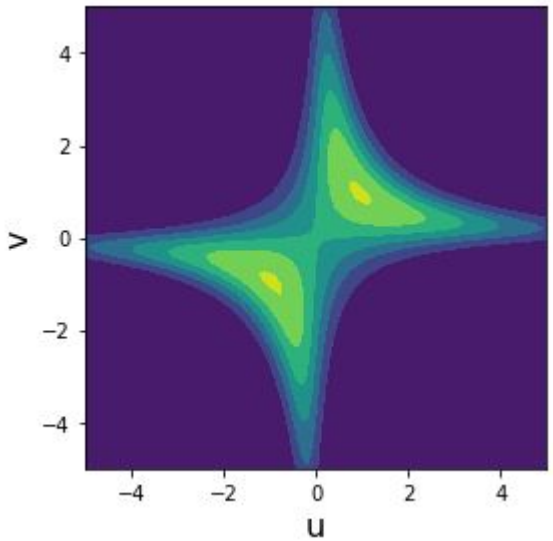
$$\frac{1}{\int \exp\Big(\mathbb{E}_{q(v)}\Big[\log p(u, v|r)\Big]\Big)\,\mathrm{d}u} = \exp(-\lambda - 1)$$

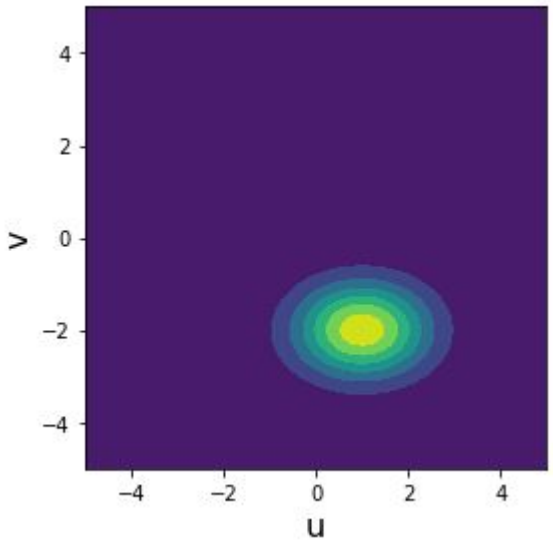
Approximation, maximum a posteriori, etc.



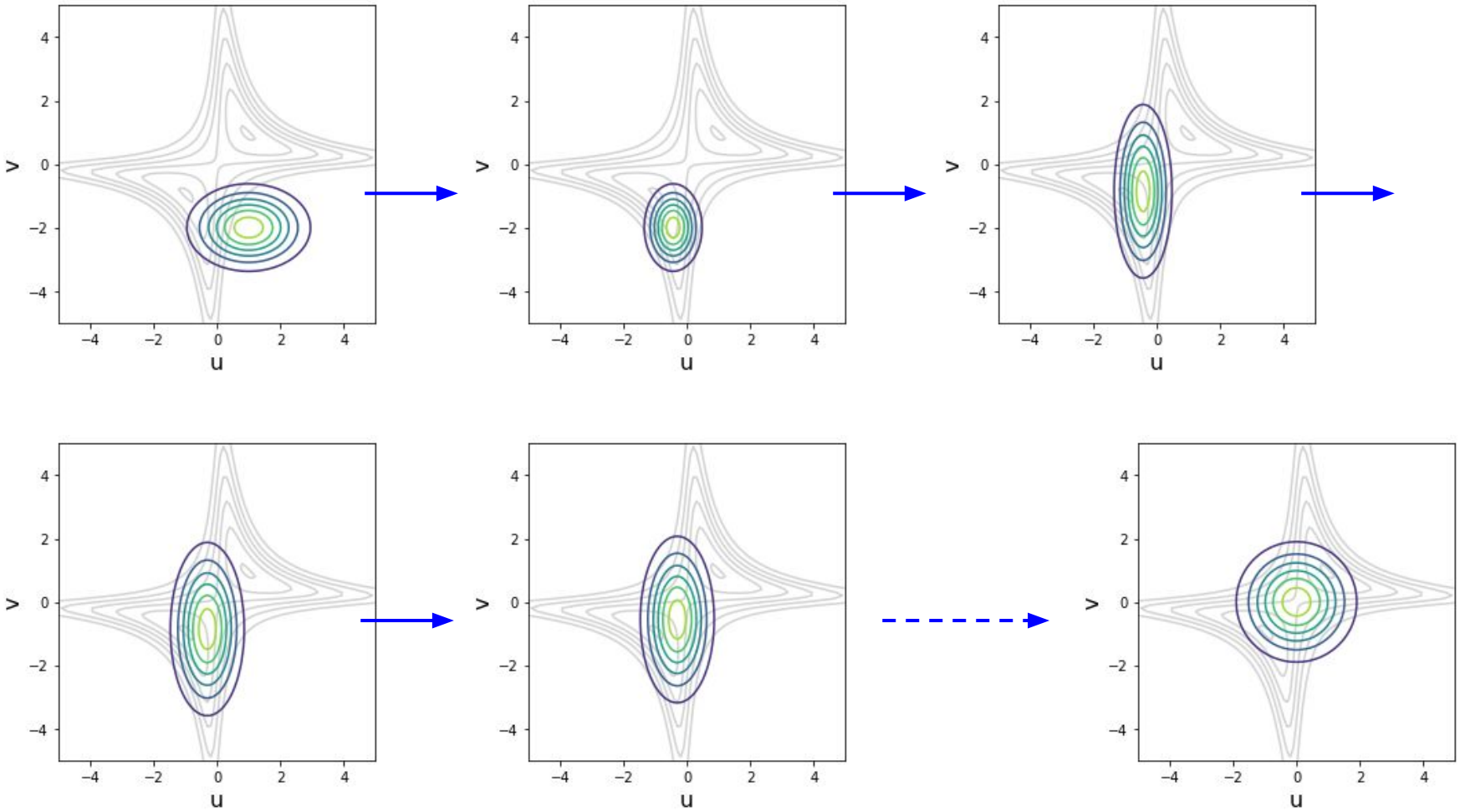
A variational approximation

$$p(u, v|r) \approx q(u)\, q(v)$$

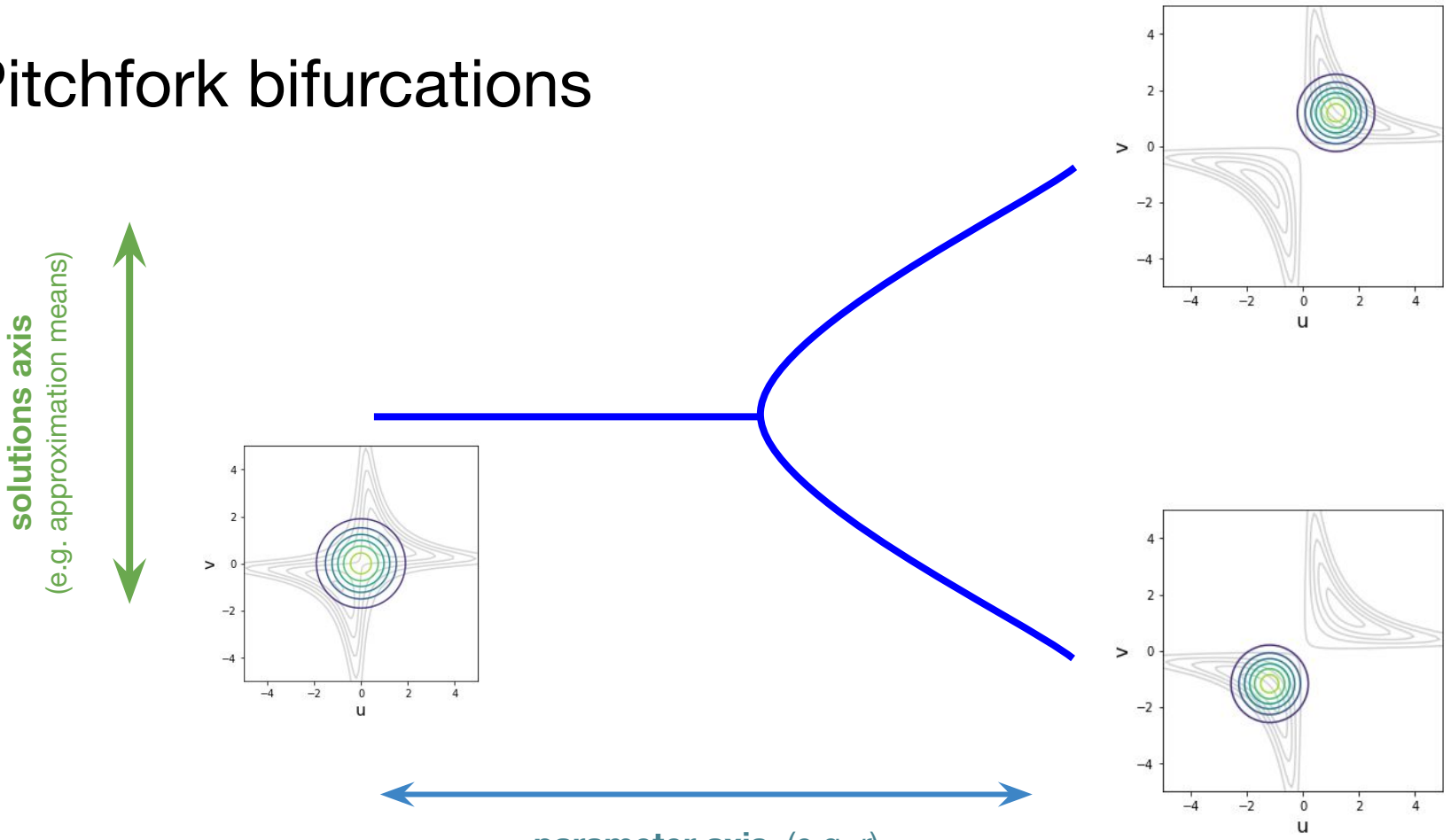




$$q(u) = \mathcal{N}(u; \mu_u, \sigma_u^2)$$
$$q(v) = \mathcal{N}(v; \mu_v, \sigma_v^2)$$


v
u
4
2
0
−2
−4
−4
−2
0
2
4

Pitchfork bifurcations



A variational update

$$q(u) \propto \exp\left(\mathbb{E}_{q(v)}\left[\log p(u, v|r)\right]\right)$$

A variational update

$$
\begin{aligned}
q(u) &\propto \exp\left(\mathbb{E}_{q(v)}\left[\log p(u, v|r)\right]\right) \\
&\propto \exp\left(\mathbb{E}_{q(v)}\left[-\tfrac{\lambda}{2}(r^2 - ruv + u^2v^2) - \tfrac{\tau}{2}u^2 + \cdots\right]\right)
\end{aligned}
$$

A variational update

$$
\begin{aligned}
q(u) &\propto \exp\left(\mathbb{E}_{q(v)}\Big[\log p(u, v|r)\Big]\right) \\
&\propto \exp\left(\mathbb{E}_{q(v)}\Big[-\tfrac{\lambda}{2}(r^2 - ruv + u^2v^2) - \tfrac{\tau}{2}u^2 + \cdots\Big]\right) \\
&\propto \exp\left(-\tfrac{\lambda}{2}\Big(r^2 - ru\,\mathbb{E}_{q(v)}[v] + u^2\,\mathbb{E}_{q(v)}[v^2]\Big) - \tfrac{\tau}{2}u^2 + \cdots\right)
\end{aligned}
$$

A variational update

$$
\begin{aligned}
q(u) &\propto \exp\left(\mathbb{E}_{q(v)}\Big[\log p(u, v|r)\Big]\right) \\
&\propto \exp\left(\mathbb{E}_{q(v)}\Big[-\tfrac{\lambda}{2}(r^2 - ruv + u^2v^2) - \tfrac{\tau}{2}u^2 + \cdots\Big]\right) \\
&\propto \exp\left(-\tfrac{\lambda}{2}\Big(r^2 - ru\,\mathbb{E}_{q(v)}[v] + u^2\,\mathbb{E}_{q(v)}[v^2]\Big) - \tfrac{\tau}{2}u^2 + \cdots\right) \\
&\propto \exp\left(-\tfrac{1}{2}\Big(\lambda\mathbb{E}_{q(v)}[v^2] + \tau\Big)u^2 + \lambda r\mathbb{E}_{q(v)}[v]u + \cdots\right)
\end{aligned}
$$

A variational update

$$
\begin{aligned}
q(u) &\propto \exp\left(\mathbb{E}_{q(v)}\Big[\log p(u,v|r)\Big]\right) \\
&\propto \exp\left(\mathbb{E}_{q(v)}\Big[-\tfrac{\lambda}{2}(r^2 - ruv + u^2v^2) - \tfrac{\tau}{2}u^2 + \cdots\Big]\right) \\
&\propto \exp\left(-\tfrac{\lambda}{2}\Big(r^2 - ru\,\mathbb{E}_{q(v)}[v] + u^2\,\mathbb{E}_{q(v)}[v^2]\Big) - \tfrac{\tau}{2}u^2 + \cdots\right) \\
&\propto \exp\left(-\tfrac{1}{2}\Big(\lambda\mathbb{E}_{q(v)}[v^2] + \tau\Big)u^2 + \lambda r\mathbb{E}_{q(v)}[v]u + \cdots\right) \\
&\propto \exp\left(-\tfrac{1}{2}\Big(\lambda\mathbb{E}_{q(v)}[v^2] + \tau\Big)\left(u - \Big(\lambda\mathbb{E}_{q(v)}[v^2] + \tau\Big)^{-1}\Big(\lambda r\mathbb{E}_{q(v)}[v]\Big)\right)^2 + \cdots\right)
\end{aligned}
$$

A variational update

$$
\begin{aligned}
q(u) &\propto \exp\left(\mathbb{E}_{q(v)}\Big[\log p(u,v|r)\Big]\right) \\
&\propto \exp\left(\mathbb{E}_{q(v)}\Big[-\tfrac{\lambda}{2}(r^2 - ruv + u^2v^2) - \tfrac{\tau}{2}u^2 + \cdots\Big]\right) \\
&\propto \exp\Big(-\tfrac{\lambda}{2}\Big(r^2 - ru\,\mathbb{E}_{q(v)}[v] + u^2\,\mathbb{E}_{q(v)}[v^2]\Big) - \tfrac{\tau}{2}u^2 + \cdots\Big) \\
&\propto \exp\Big(-\tfrac{1}{2}\Big(\lambda\mathbb{E}_{q(v)}[v^2] + \tau\Big)u^2 + \lambda r\mathbb{E}_{q(v)}[v]u + \cdots\Big) \\
&\propto \exp\left(-\tfrac{1}{2}\Big(\lambda\mathbb{E}_{q(v)}[v^2] + \tau\Big)\left(u - \Big(\lambda\mathbb{E}_{q(v)}[v^2] + \tau\Big)^{-1}\Big(\lambda r\mathbb{E}_{q(v)}[v]\Big)\right)^2 + \cdots\right) \\
q(u) &= \mathcal{N}\left(u\,;\, \underbrace{\Big(\lambda\mathbb{E}_{q(v)}[v^2] + \tau\Big)^{-1}\Big(\lambda\mathbb{E}_{q(v)}[v]r\Big)}_{\text{mean}},\, \underbrace{\Big(\lambda\mathbb{E}_{q(v)}[v^2] + \tau\Big)^{-1}}_{\text{variance}}\right)
\end{aligned}
$$

A variational update

$$q(u) = \mathcal{N}\left(u\,;\, \underbrace{\left(\lambda \mathbb{E}_{q(v)}[v^2] + \tau\right)^{-1}\left(\lambda \mathbb{E}_{q(v)}[v] r\right)}_{\text{mean}},\, \underbrace{\left(\lambda \mathbb{E}_{q(v)}[v^2] + \tau\right)^{-1}}_{\text{variance}}\right)$$

$$q(u) = \mathcal{N}\left(u\,;\, \underbrace{\left(\lambda(\sigma_v^2 + \mu_v^2) + \tau\right)^{-1}\left(\lambda \mu_v r\right)}_{\mu_u},\, \underbrace{\left(\lambda(\sigma_v^2 + \mu_v^2) + \tau\right)^{-1}}_{\sigma_u^2}\right)$$

A variational update

$$q(u) = \mathcal{N}\left(u\,;\, \underbrace{\left(\lambda \mathbb{E}_{q(v)}[v^2] + \tau\right)^{-1}\left(\lambda \mathbb{E}_{q(v)}[v] r\right)}_{\text{mean}},\, \underbrace{\left(\lambda \mathbb{E}_{q(v)}[v^2] + \tau\right)^{-1}}_{\text{variance}}\right)$$

$$q(u) = \mathcal{N}\left(u\,;\, \underbrace{\left(\lambda(\sigma_v^2 + \mu_v^2) + \tau\right)^{-1}\left(\lambda \mu_v r\right)}_{\mu_u},\, \underbrace{\left(\lambda(\sigma_v^2 + \mu_v^2) + \tau\right)^{-1}}_{\sigma_u^2}\right)$$

Compare to

$$\mathbf{u}_m = \left(\lambda \sum_{n \in \Omega(m)} \mathbf{v}_n \mathbf{v}_n^T + \tau \mathbf{I}\right)^{-1} \left(\lambda \sum_{n \in \Omega(m)} r_{mn} \mathbf{v}_n\right)$$

A variational update

$$q(u) = \mathcal{N}\left(u\,;\, \underbrace{\left(\lambda \mathbb{E}_{q(v)}[v^2] + \tau\right)^{-1}\left(\lambda \mathbb{E}_{q(v)}[v] r\right)}_{\text{mean}},\, \underbrace{\left(\lambda \mathbb{E}_{q(v)}[v^2] + \tau\right)^{-1}}_{\text{variance}}\right)$$

$$q(u) = \mathcal{N}\left(u\,;\, \underbrace{\left(\lambda(\sigma_v^2 + \mu_v^2) + \tau\right)^{-1}\left(\lambda \mu_v r\right)}_{\mu_u},\, \underbrace{\left(\lambda(\sigma_v^2 + \mu_v^2) + \tau\right)^{-1}}_{\sigma_u^2}\right)$$

Compare to

$$p(u|r,v) = \frac{p(r|u,v)\,p(u)}{p(r|v)}$$
$$= \mathcal{N}(u\,;\, (\lambda v^2 + \tau)^{-1}(\lambda r v),\, (\lambda v^2 + \tau)^{-1})$$

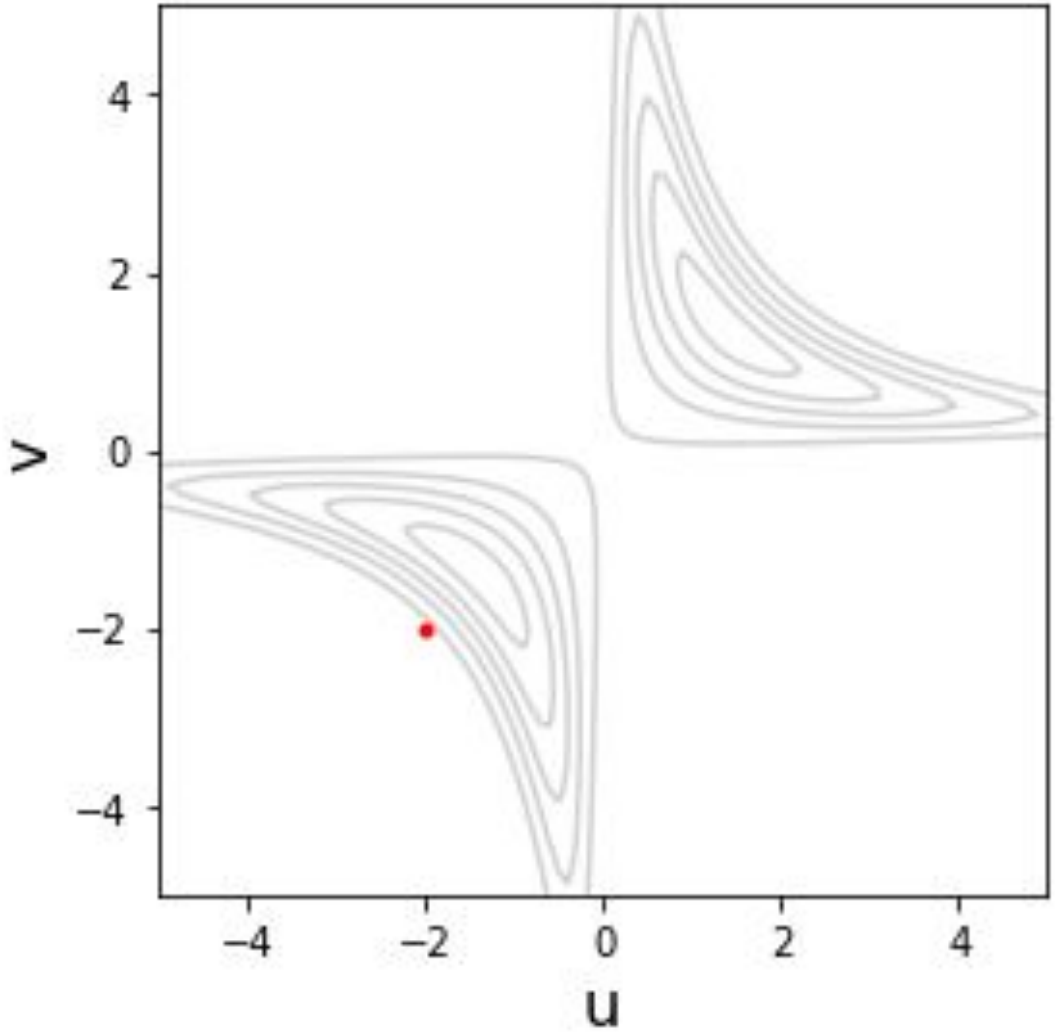
Gibbs sampling (Markov chain Monte Carlo)

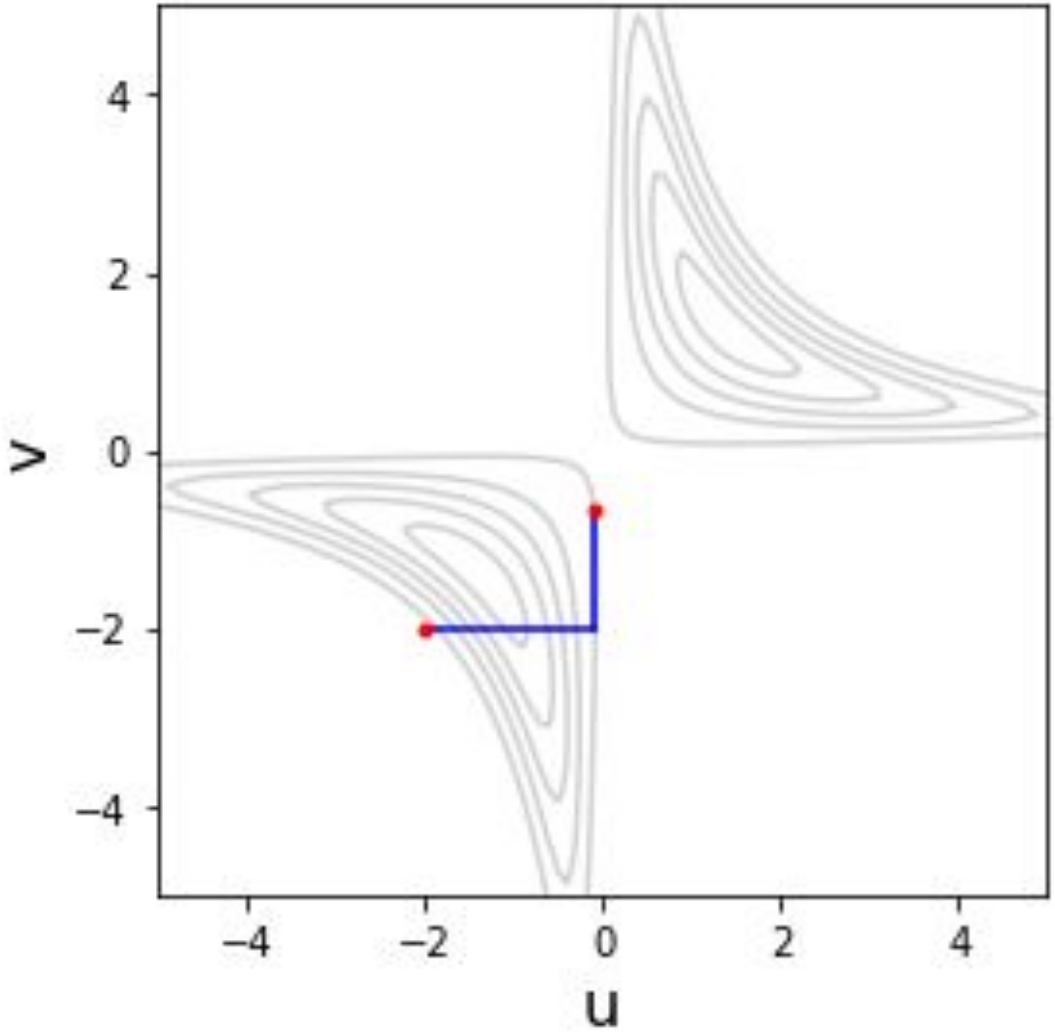
Repeat:

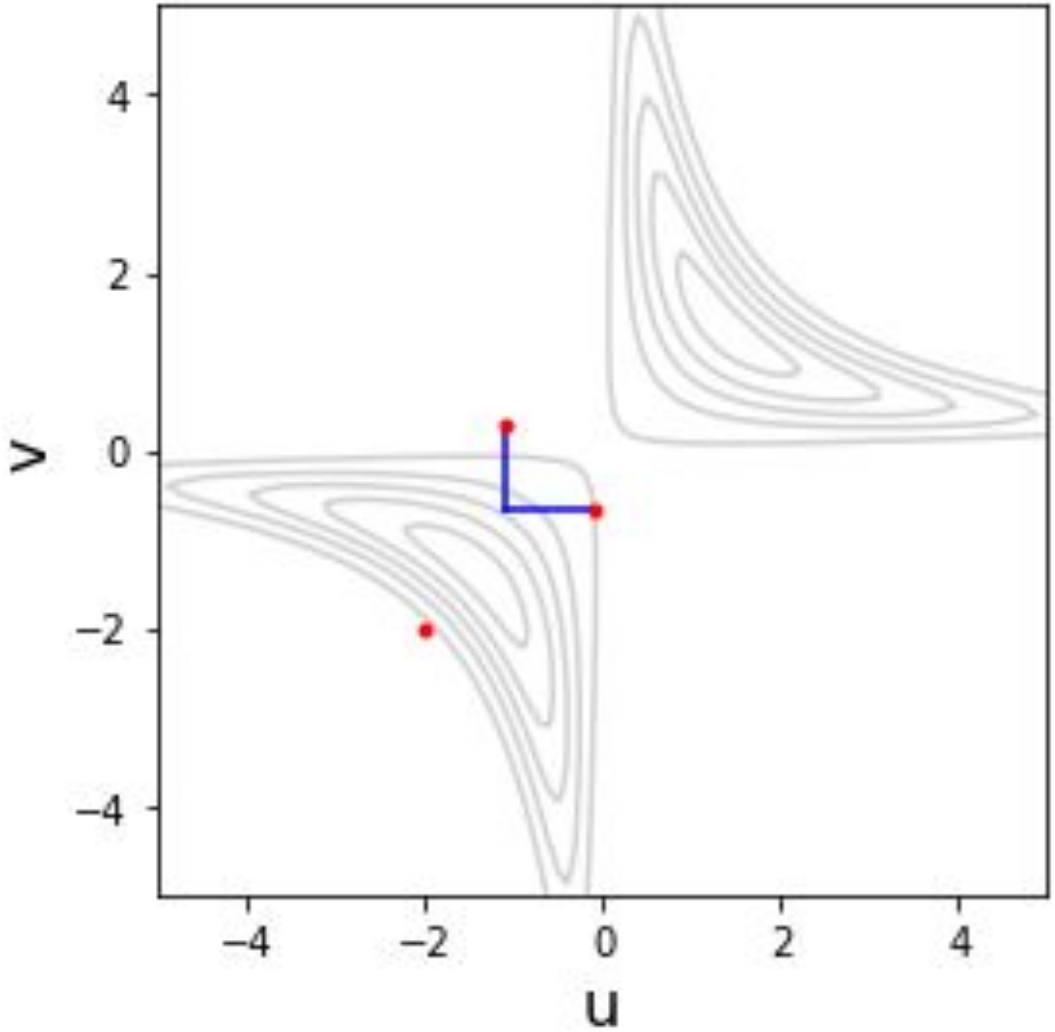
Sample $u \sim p(u|r, v)$

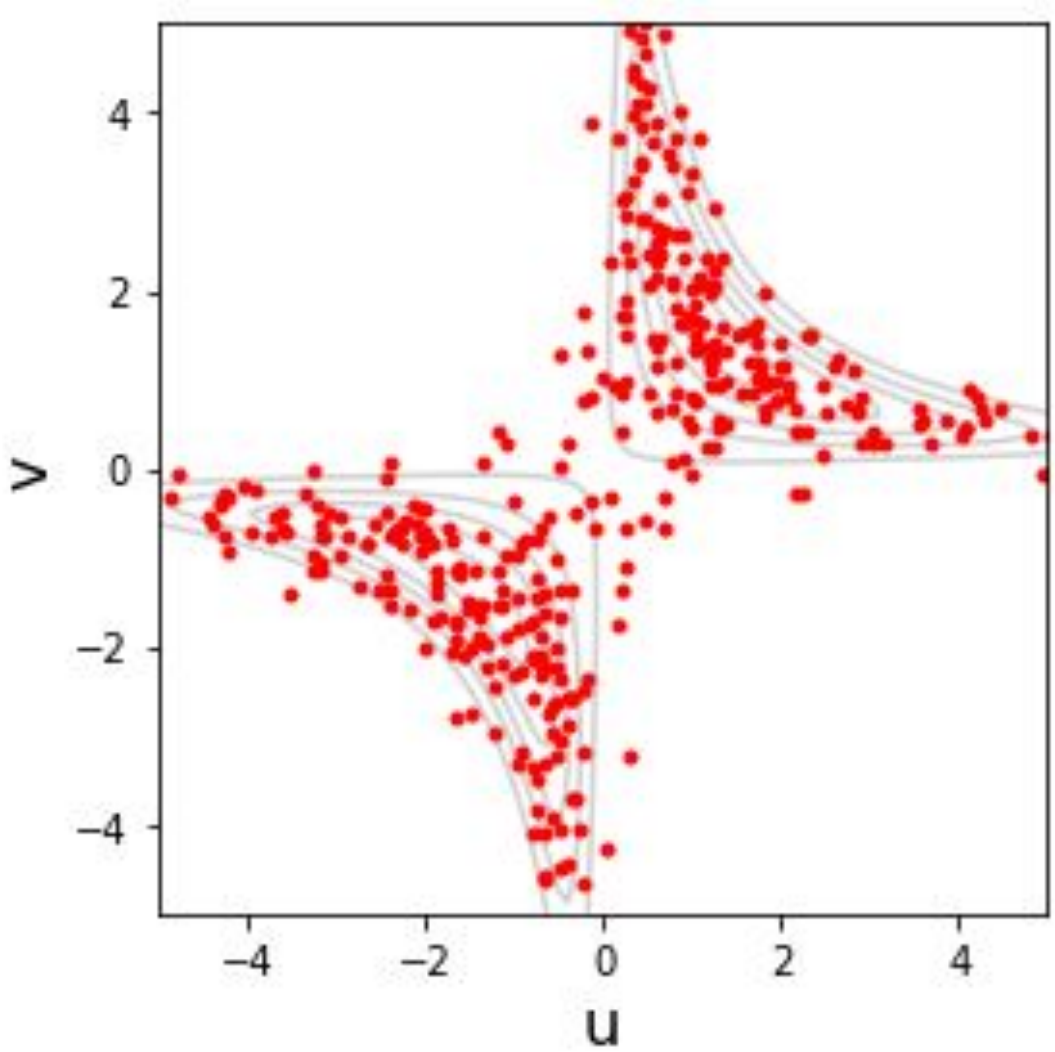
Sample $v \sim p(v|r, u)$

Use samples in **empirical average**


4
2
0
−2
−4
v
−4
−2
0
2
4
u


4
2
0
−2
−4
v
−4
−2
0
2
4
u


4
2
0
−2
−4
v
−4
−2
0
2
4
u


4
2
0
−2
−4
v
−4
−2
0
2
4
u

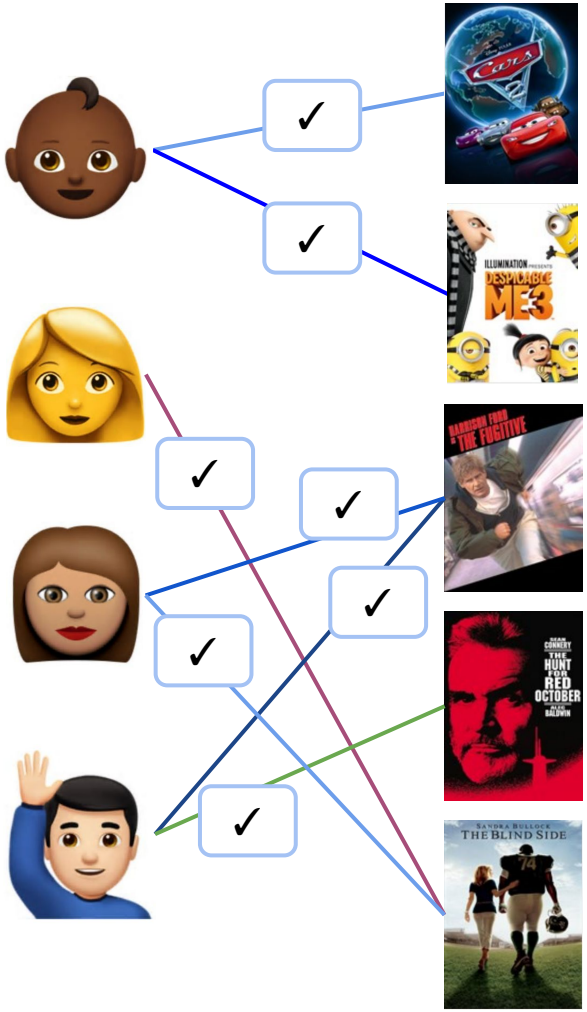
Going back to a production system…

Implicit feedback

Practical point

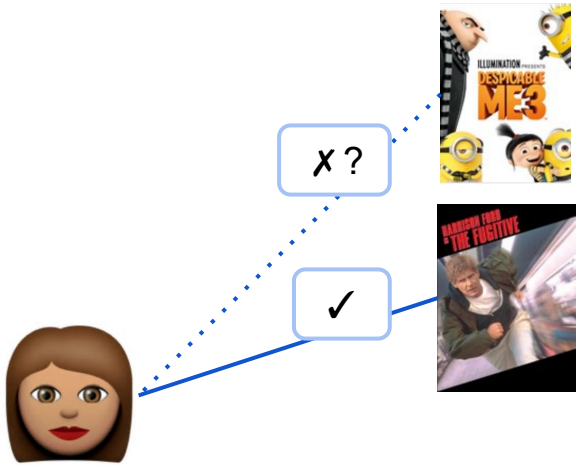
Implicit feedback

“One-class collaborative filtering”



There are many other signals (count; temporal; contextual features) which we’ll ignore until later...

Practical point

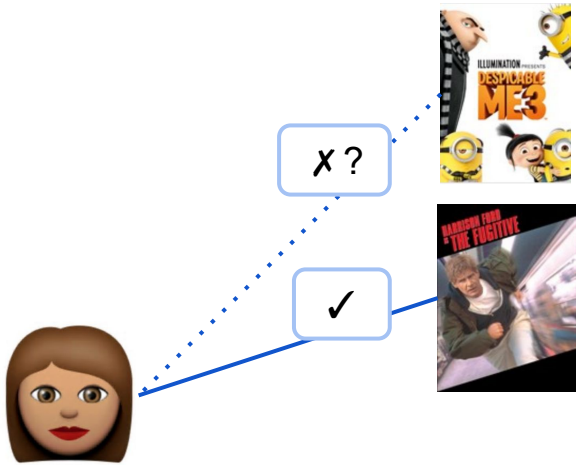


$$p(r_{mn} = 1 | \mathbf{u}_m, \mathbf{v}_n) = \text{sigmoid}(\mathbf{u}_m^T \mathbf{v}_n)$$

$$p(r_{mn} = 0 | \mathbf{u}_m, \mathbf{v}_n) = 1 - \text{sigmoid}(\mathbf{u}_m^T \mathbf{v}_n)$$

$$\text{sigmoid}(a) = \frac{1}{1 + \mathrm{e}^{-a}}$$

Practical point

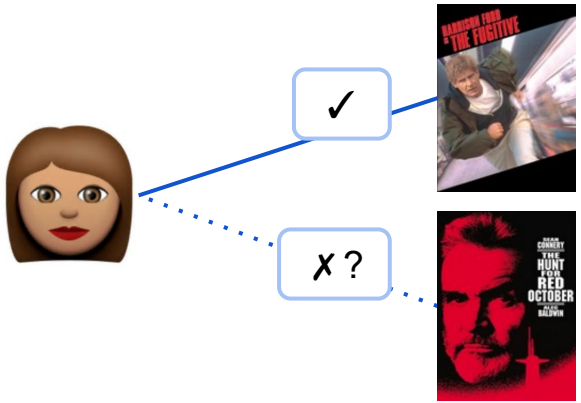


Random unobserved “negative” **graph**: the content that a user dislikes

(We write the likelihood as an expectation over all possible negative graphs, where the “negative graph” follows the same power law distribution as the observed graph)

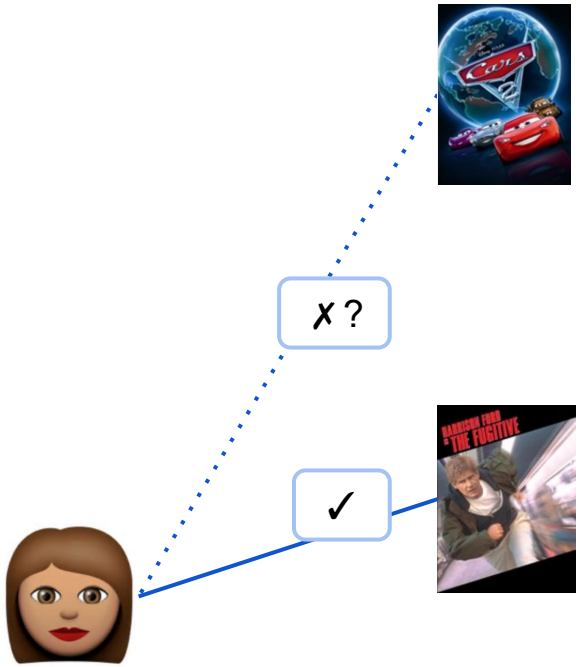
Stochastic gradient descent over samples from such graphs

Practical point



Stochastic gradient descent step 1

Practical point

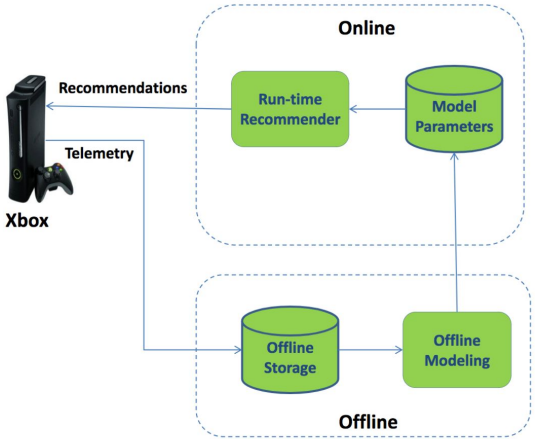


Stochastic gradient descent step 2

etc...

A basic version is (almost) production-ready!

- In the *first-cut* version, we only need to
  - store a mean and variance vector for each user, plus biases
  - store a mean and variance vector for each item, plus biases
  - load a user's vector at runtime
  - compute predictions over all items
  - ...and return something (like the top)



One-class Collaborative Filtering with Random Graphs
Paquet + Koenigstein, WWW 2013

Adding Features

Better priors

[adventure, animated, boys, cars, kids]
[0 1 1 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0]

[adventure, animated, anti-hero, kids]
[0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

If we have features for some content, and there might be some correlation between features and usage patterns:

- Learn which features are useful for predictions

Hierarchical Bayesian model:

prior → items → usage ← user ← prior

prior → features → items → usage ← user ← prior

Better priors

Hierarchical prior:

[adventure, animated, boys, cars, kids]
[0 1 1 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0]

[adventure, animated, anti-hero, kids]
[0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

p(v|F)

$$p(\mathbf{v}_n) = \mathcal{N}\left(\mathbf{v}_n\,;\, \frac{1}{\sqrt{F_n}} \sum_{i \in \text{features}(n)} \mathbf{f}_i,\, \tau\mathbf{I}\right)$$

$$p(\mathbf{f}_i) = \mathcal{N}\left(\mathbf{f}_i\,;\, \mathbf{0},\, \beta\mathbf{I}\right)$$

Adding features: loss function without features

$$\mathcal{L}(\mathbf{U}, \mathbf{V}, \mathbf{b}^{(u)}, \mathbf{b}^{(i)}) = -\frac{\lambda}{2} \sum_{m=1}^{M} \sum_{n \in \Omega(m)} \left(r_{mn} - (\mathbf{u}_m^T \mathbf{v}_n + b_m^{(u)} + b_n^{(i)}) \right)^2$$

$$- \frac{\tau}{2} \sum_{m=1}^{M} (\mathbf{v}_n - \mathbf{0})^T (\mathbf{v}_n - \mathbf{0})$$

$$- \frac{\tau}{2} \sum_{m=1}^{M} \mathbf{u}_m^T \mathbf{u}_m$$

$$- \frac{\tau_{\text{bias}}}{2} \sum_{m=1}^{M} (b_m^{(u)})^2 - \frac{\tau_{\text{bias}}}{2} \sum_{n=1}^{N} (b_n^{(i)})^2$$

Adding features: loss function with features

$$\mathcal{L}(\mathbf{U}, \mathbf{V}, \mathbf{b}^{(u)}, \mathbf{b}^{(i)}, \mathbf{F}) = -\frac{\lambda}{2} \sum_{m=1}^{M} \sum_{n \in \Omega(m)} \left(r_{mn} - (\mathbf{u}_m^T \mathbf{v}_n + b_m^{(u)} + b_n^{(i)}) \right)^2$$
$$- \frac{\tau}{2} \sum_{n=1}^{N} \left(\mathbf{v}_n - \frac{1}{\sqrt{F_n}} \sum_{\iota \in \text{features}(n)} \mathbf{f}_\iota \right)^T \left(\mathbf{v}_n - \frac{1}{\sqrt{F_n}} \sum_{\iota \in \text{features}(n)} \mathbf{f}_\iota \right)$$
$$- \frac{\tau}{2} \sum_{m=1}^{M} \mathbf{u}_m^T \mathbf{u}_m - \frac{\tau}{2} \sum_{\iota=1}^{\text{num feats}} \mathbf{f}_\iota^T \mathbf{f}_\iota$$
$$- \frac{\tau_{\text{bias}}}{2} \sum_{m=1}^{M} (b_m^{(u)})^2 - \frac{\tau_{\text{bias}}}{2} \sum_{n=1}^{N} (b_n^{(i)})^2$$

Adding features: loss function with features

$$\mathcal{L}(\mathbf{U}, \mathbf{V}, \mathbf{b}^{(u)}, \mathbf{b}^{(i)}, \mathbf{F}) = -\frac{\lambda}{2} \sum_{m=1}^{M} \sum_{n \in \Omega(m)} \left(r_{mn} - (\mathbf{u}_m^T \mathbf{v}_n + b_m^{(u)} + b_n^{(i)}) \right)^2$$

$$- \frac{\tau}{2} \sum_{n=1}^{N} \left(\mathbf{v}_n - \frac{1}{\sqrt{F_n}} \sum_{\iota \in \text{features}(n)} \mathbf{f}_\iota \right)^T \left(\mathbf{v}_n - \frac{1}{\sqrt{F_n}} \sum_{\iota \in \text{features}(n)} \mathbf{f}_\iota \right)$$

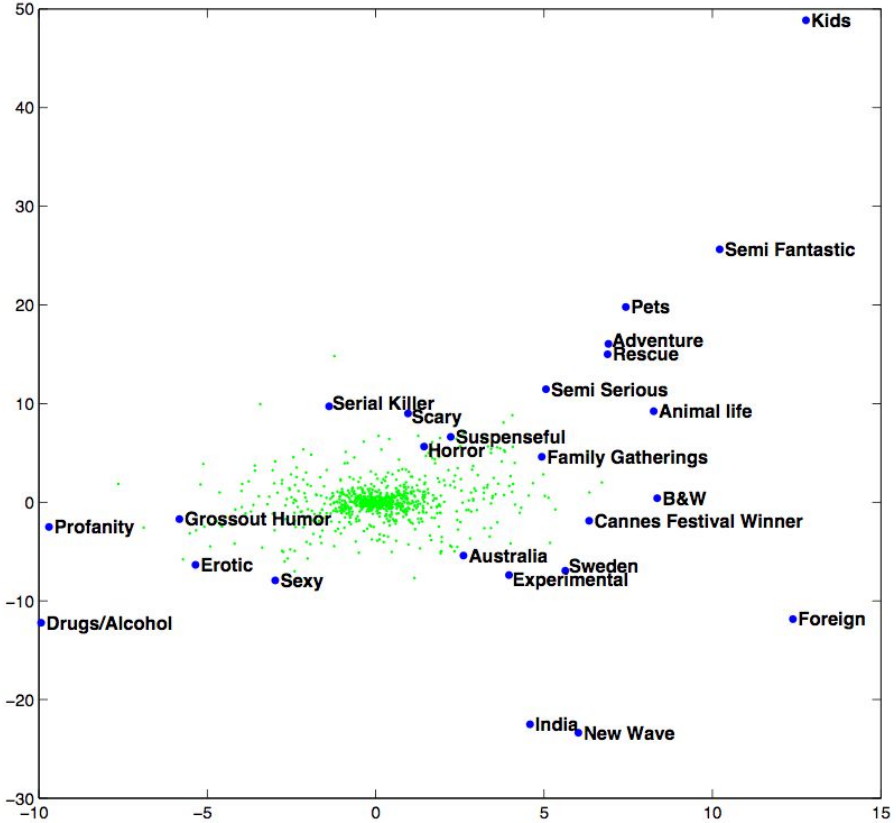
$$- \frac{\tau}{2} \sum_{m=1}^{M} \mathbf{u}_m^T \mathbf{u}_m - \frac{\tau}{2} \sum_{\iota=1}^{\text{num feats}} \mathbf{f}_\iota^T \mathbf{f}_\iota$$

$$- \frac{\tau_{\text{bias}}}{2} \sum_{m=1}^{M} (b_m^{(u)})^2 - \frac{\tau_{\text{bias}}}{2} \sum_{n=1}^{N} (b_n^{(i)})^2$$

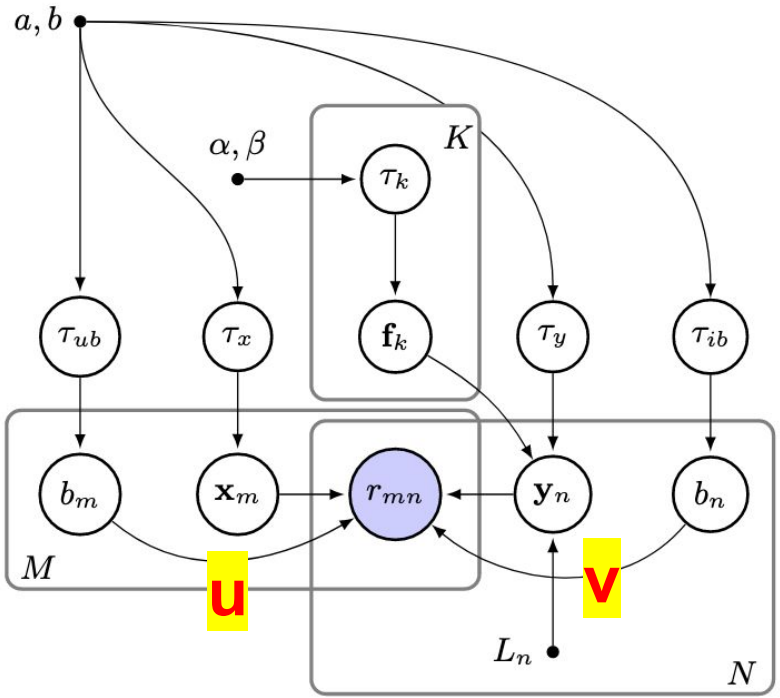
Scratch space

Posterior means for each feature

This shares prior information, and helps with the "**cold start**" problem...



Probabilistic graphical model



Practical 5

Adding features

Practical 5 | **optional extra**

What to do with **side information** and **additional features**?

✅ Build genre information like

Dataset examples

movies.csv

```
movieId,title,genres
1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy
2,Jumanji (1995),Adventure|Children|Fantasy
3,Grumpier Old Men (1995),Comedy|Romance
```

into your Alternating Least Squares (explicit feedback) or Bayesian Personalized Ranking (implicit feedback) model

Ideas discussed in session are from the XBox system

✅ How does it help the **cold start problem**? (When a movie has no ratings, but you still want to predict it)

Practical 5 | Code with user, item and feature vectors

✅ Your code with **alternating least squares** will look something like this:

```
repeat:
  # update user biases and vectors
  for m in 0 .. M-1:
    ...
    for (n, r) in get_items_and_ratings_for_user(m):
      ...

  # now do the same for the items
  for n in 0 .. N-1:
    ...

  # now update feature vectors
  for i in 0 .. num_features-1:
    ...
```

Menu

Preferences

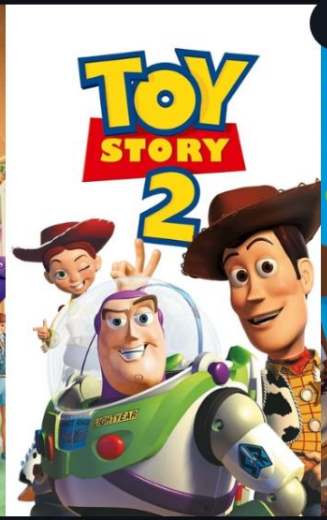
Search

Recommendation

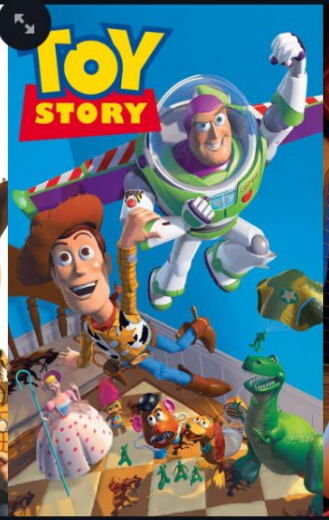
Personalisation

User could like something like this

Toy Story 3 (2010)
Adventure|Animation|Children|Comedy|Fantasy|IMAX

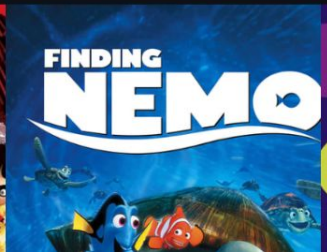


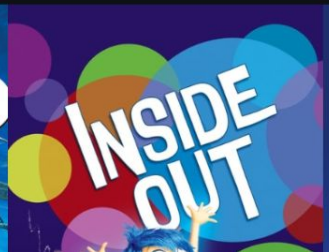
Toy Story 2 (1999)
Adventure|Animation|Children|Comedy|Fantasy

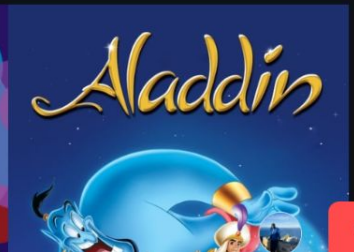


Toy Story (1995)
Adventure|Animation|Children|Comedy|Fantasy

Beauty and the Beast (1991)
Animation|Children|Fantasy|Musical|Romance|IMAX







Menu

Preferences

Search

Recommendation

Popular

Personalisation

tim could like something like this

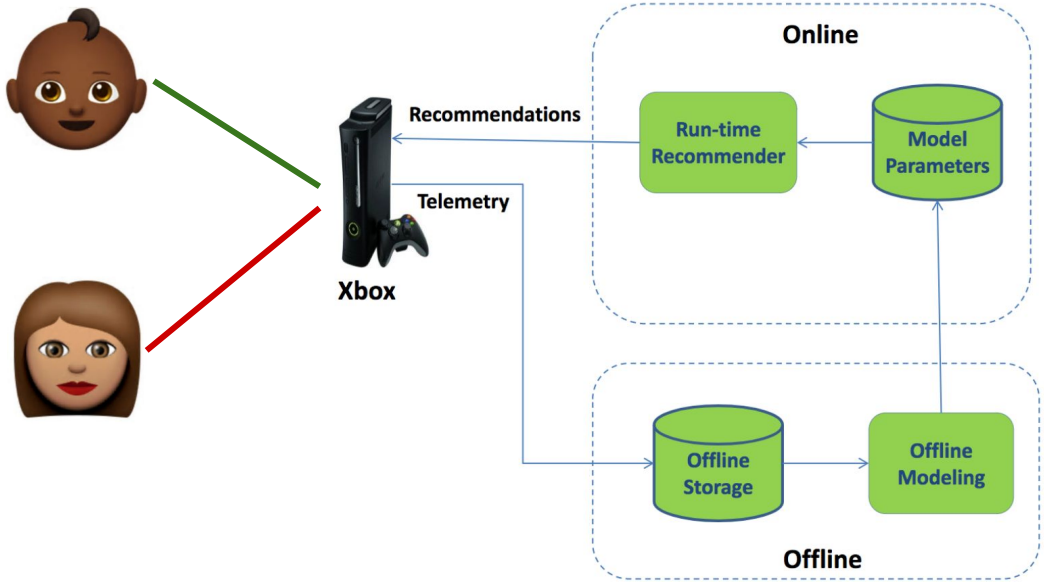
Spider-Man 2 (2004) Action|Adventure|Sci-Fi|IMAX

Avengers, The (2012) Action|Adventure|Sci-Fi|IMAX

Captain America: The Winter Soldier (2014) Action|Adventure|Sci-Fi|IMAX

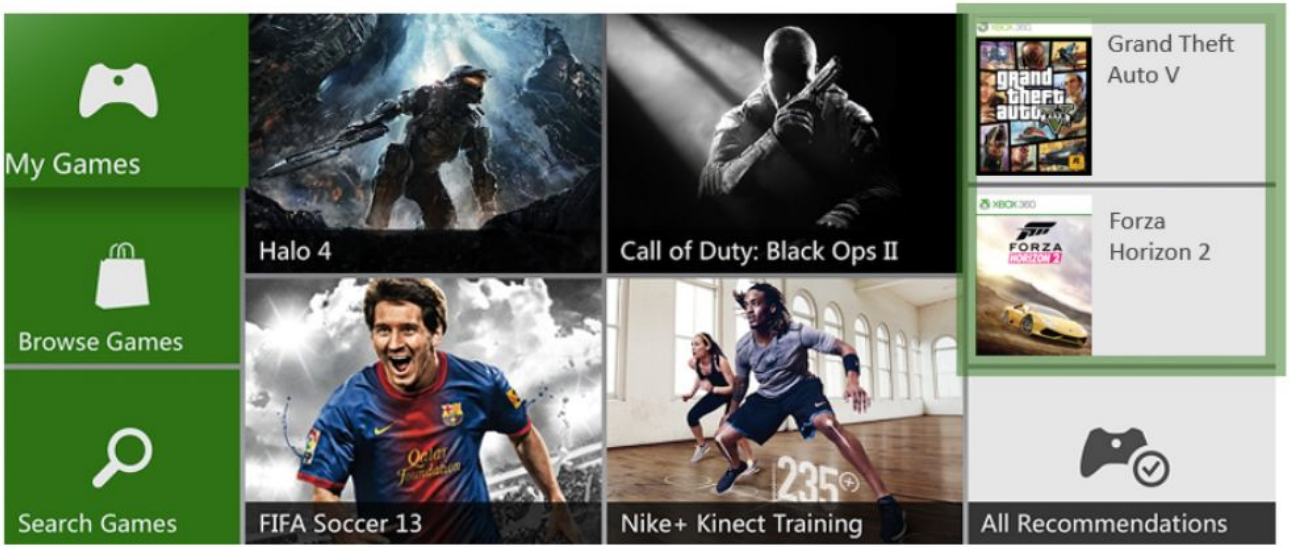
Avengers: Age of Ultron (2015) Action|Adventure|Sci-Fi

Mixture models and multiple users



Optimizing for a slate

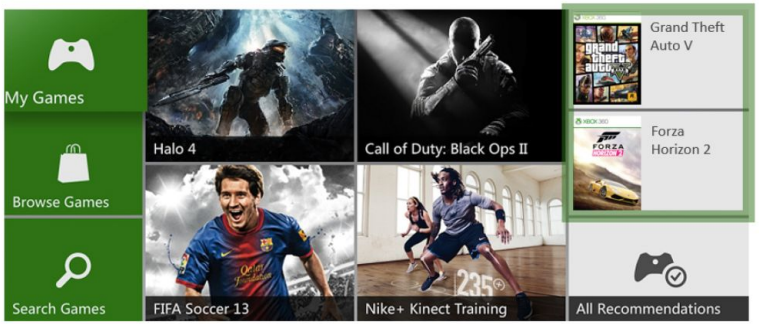
Xbox 360, some time ago...

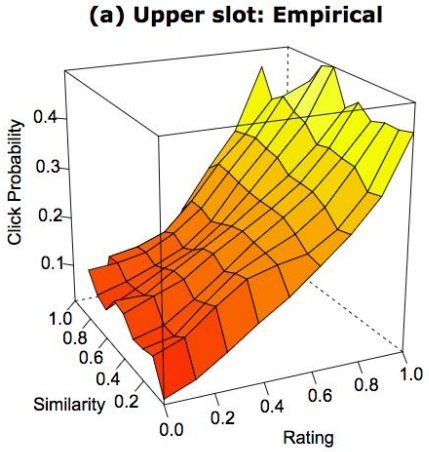


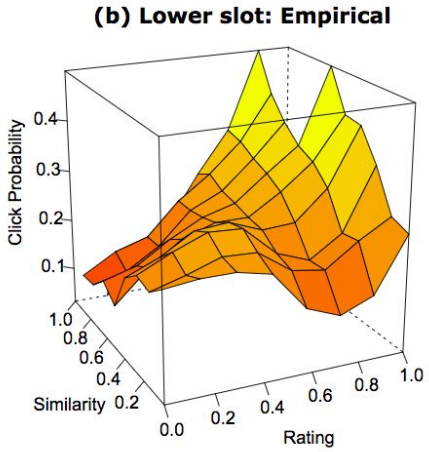
Optimizing for a slate

Uncertainty counts! If we're not certain if someone is going to click on an item in a slate, it is better to **diversify**…

But if we are certain, then we place two similar items







Implicit feedback: ranking problem!

Practical 6

Ranking

Practical 6 | **Reading list**

Read:

✅ BPR: Bayesian Personalized Ranking from Implicit Feedback

✅ One-class Collaborative Filtering with Random Graphs

✅ Dive into Deep Learning, chapter 16.5

Practical 6

✅ Assume that the data **implicit feedback data** (your own conversion)

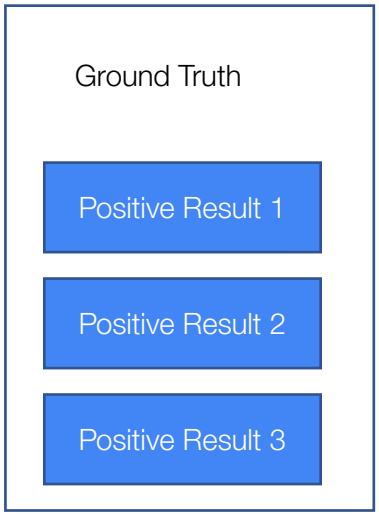
✅ Code BPR (Bayesian personalised ranking) using the SGD framework

✅ Repeat practical 1 on Movielens, but with **stochastic gradient descent** instead of alternating least squares

✅ Discuss and analyse of differences (with alternating least squares)

Practical 6

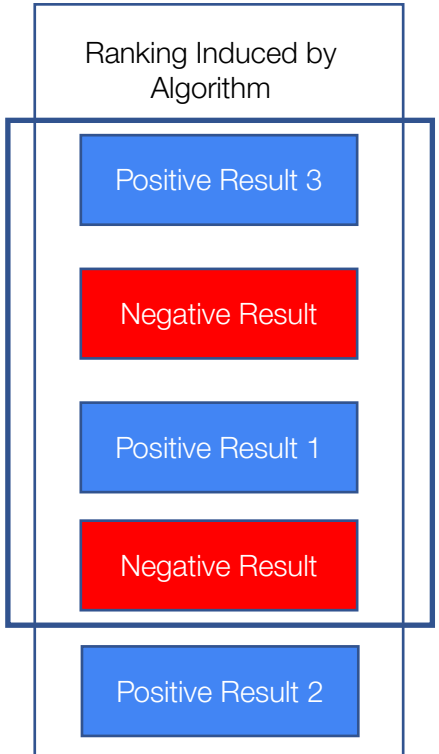
✅ Use **precision@k** and **recall@k** to show that your BPR algorithm works



$$\boldsymbol{k = 4}$$

$$\boldsymbol{precision@k} = \frac{\#tp}{\#tp + \#fp} = \frac{2}{2+2} = \frac{1}{2}$$

$$\boldsymbol{recall@k} = \frac{\#tp}{\#tp + \#fn} = \frac{2}{2+1} = \frac{2}{3}$$



Bird’s eye

Feedback types

Explicit feedback

- 5-star scale (e.g., Netflix)
- 0-100 scale (e.g., Yahoo! Music)
- Binary- thumb up/down

Implicit feedback

- Watching a movie
- Purchasing an app
- Clicking on an article
- Skipping a song (negative?)

Issues with explicit signals (1)

Explicit signals (usually) do not exist

- Many systems do not have an interface for ratings
- Even when a rating system is in place, there is often much more implicit feedback than explicit ratings
- Ratings of the same movie can change when the order of ratings change

Issues with explicit signals (2)

Missing not at random!

- People rate what they watched
- If it's not in your "taste" you will not care to rate it
- Negative values are still a positive indication (because the fact that you watched a movie is a positive indication even if you gave it a low rating)
- See Benjamin Marlin et al., Collaborative Filtering and the Missing at Random Assumption

Issues with explicit signals (3)

Context matters

- Ratings of the same movies change when the order of the ratings change
  For example, if the first movie is great, the entire session might be rated lower than if the first movie is terrible

- Users change their perspectives after some time
  For example, movie reviews and perception influence user ratings
  A bias can occur between users who rated immediately after watching when the movie was released and users who rate long after watching a movie

Issues with explicit signals (4)

Much of the signal is “hidden” in biases

For example in the Netflix dataset as well as in Yahoo! Music there is more signal in modeling user and item biases than in the actual user-item personalization signal

R^2 in explicit RMSE-based collaborative filtering datasets

R^2 coefficient of determination

One minus the ratio of the model's MSE to the total variance in the dataset

R^2 = 1 - MSE / Var

A simple model that predicts the mean rating value achieves $R^2 = 0$

Positive values indicate an improvement over the mean baseline

Negative values indicate that the model is "garbage"

R^2 in explicit RMSE-based CF datasets

Biases are the main factor in explaining a user's rating. No personalization gets you far!

Netflix Prize:

total variance: 1.276
winning model: 0.73 (RMSE of 0.85)
R^2 = 0.43 of which 0.33 are attributed to biases alone

Yahoo! Music (for KDD Cup '11):

total variance: 1084.5
final model: 510.3 (RMSE of 22.59)
R^2 = 0.53 of which 0.41 are attributed to biases alone

Yahoo! Music Recommendations: Modeling Music Ratings with Temporal Dynamics and Taxonomy
Gideon Dror, Noam Koenigstein and Yehuda Koren

Issues with implicit feedback signals (1)

No negative feedback is available implicitly

- Most of the implicit feedback is positive
  For example, if a user didn't watch a movie it doesn't mean she didn't like it
  It's a one-class problem, not a binary classification problem

Sometimes implicit signals are cumulative (listening to a song multiple times) but unlike ratings, there is no scale and no maximum value

Issues with implicit feedback signals (2)

Implicit feedback is more noisy

- Sometimes, people purchase things purely because they are on sale
- One one platform, people consume content that's not available on another platform, hence making them appear more similar

Evaluation is less straightforward

Explicit vs Implicit Feedback

Explicit Feedback	Implicit Feedback
Explicit ratings of users to items	Implicit user actions indicating interest
Examples: 5-star scale, 0-100, thumb up/down (binary)	Examples: Purchase signals, clicks, playing games or music, skipping songs, watching movies, stopping in the middle of a movie,
Rare to find	Abundant in almost any company with users
Users are inconsistent among themselves and across time	Tend to follow a more “stable” behavior
Regression models	Binary models, Ranking models, Triplet loss models, One-class models
Evaluation: RMSE, MAE, etc.	Evaluation: MPR, MRR, Hit Rate, Precision@K, Recall@K, etc
Original models (Netflix Prize, KDD 2011)	Most recent recommender systems models
Much of the work is modelling biases	More inline with actual user preferences

Prominent Implicit Feedback Papers

- Collaborative Filtering for Implicit Feedback Datasets
  Y. Hu, Y. Koren, C. Volinsky

- Implicit-to-Explicit Ordinal Logistic Regression
  D. Parra, A. Karatzoglou, X. Amatriain, I. Yavuz

- BPR - Bayesian Personalized Ranking
  S. Rendle, C. Freudenthaler, Z. Gantner, and L. S. Thieme

- RankALS – Alternating Least Squares for Personalized Ranking
  G. Takacs, D. Tikk

- CLiMF – Reciprocal Rank Optimization
  Y Shi, A. Karatzoglou, L. Baltrunas, M. Larson, N. Oliver, A. Hanjalic

Blackboard class explanation

BPR: Bayesian Personalized Ranking from Implicit Feedback

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner and Lars Schmidt-Thieme
{srendle, freudenthaler, gantner, schmidt-thieme}@ismll.de
Machine Learning Lab, University of Hildesheim
Marienburger Platz 22, 31141 Hildesheim, Germany

Abstract

Item recommendation is the task of predicting a personalized ranking on a set of items (e.g. websites, movies, products). In this paper, we investigate the most common scenario with implicit feedback (e.g. clicks, purchases). There are many methods for item recommendation from implicit feedback like matrix factorization (MF) or adaptive k-

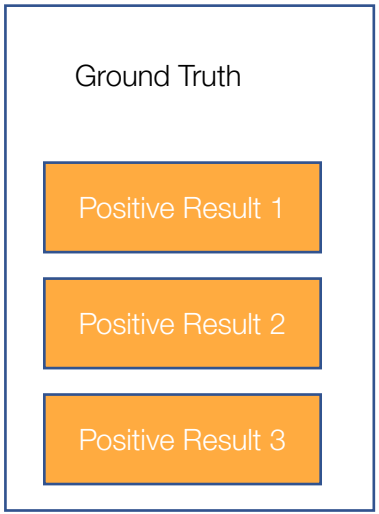
sonalization is attractive both for content providers, who can increase sales or views, and for customers, who can find interesting content more easily. In this paper, we focus on item recommendation. The task of item recommendation is to create a user-specific ranking for a set of items. Preferences of users about items are learned from the user's past interaction with the system – e.g. his buying history, viewing history, etc.

Recommender systems are an active topic of research. Most recent work is on scenarios where users provide

1: **procedure** LEARNBPR(D_S, Θ)
2: initialize Θ
3: **repeat**
4: draw (u, i, j) from D_S
5: $\Theta \leftarrow \Theta + \alpha \left(\frac{e^{-\hat{x}_{uij}}}{1+e^{-\hat{x}_{uij}}} \cdot \frac{\partial}{\partial \Theta} \hat{x}_{uij} + \lambda_\Theta \cdot \Theta \right)$
6: **until** convergence
7: **return** $\hat{\Theta}$
8: **end procedure**

Evaluation

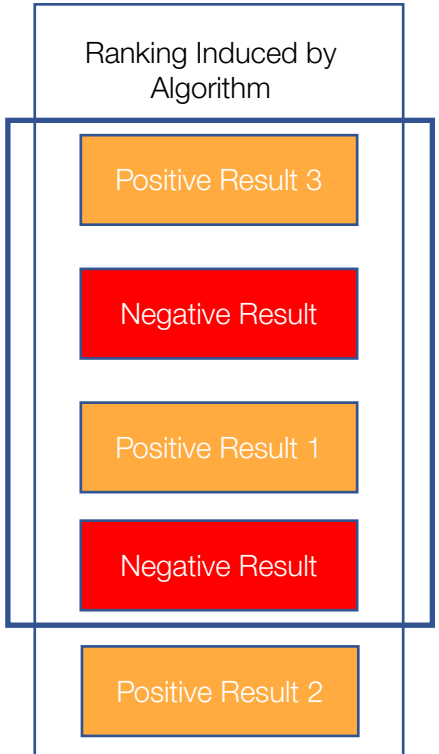
Precision@k / Recall@k



$$\boldsymbol{k = 4}$$

$$\boldsymbol{precision@k} = \frac{\#tp}{\#tp + \#fp} = \frac{2}{2+2} = \frac{1}{2}$$

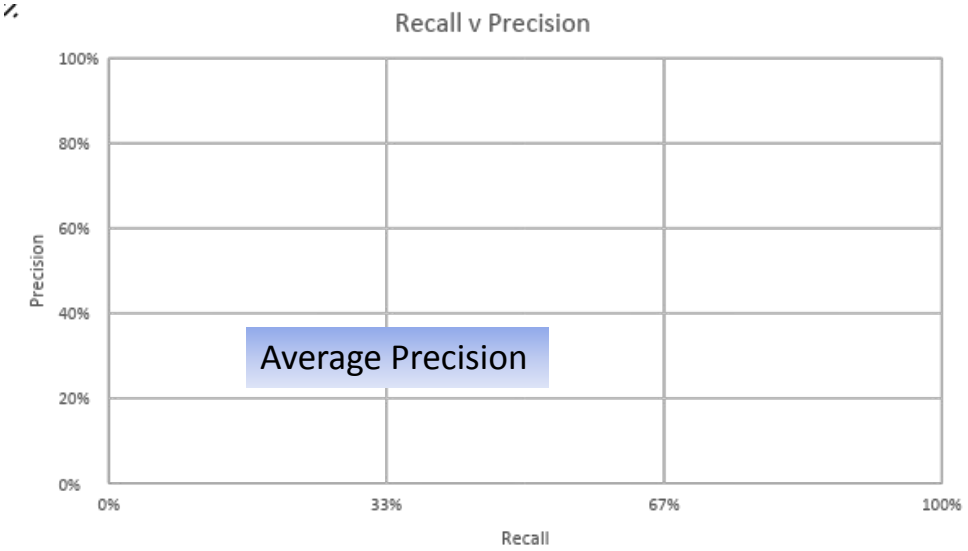
$$\boldsymbol{recall@k} = \frac{\#tp}{\#tp + \#fn} = \frac{2}{2+1} = \frac{2}{3}$$

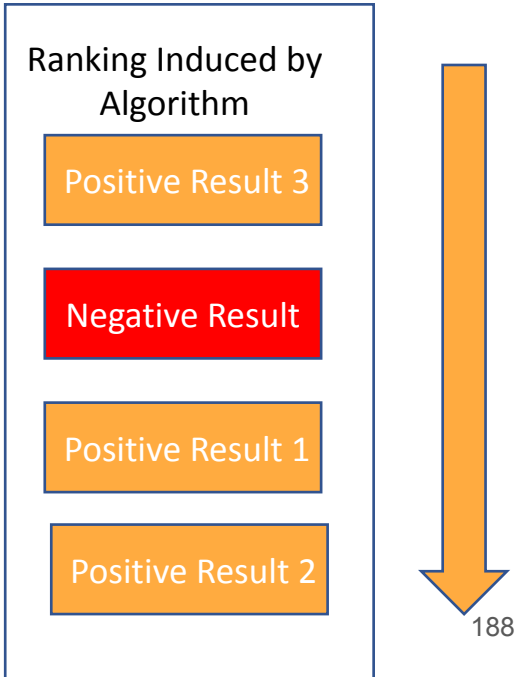


Used for: Implicit Feedback

Mean Average Precision

We can plot precision as a function of recall





NDCG@k - Normalized Discounted Cumulative Gain

The relevance is discounted by $\gamma_i = \frac{1}{\log_2(i+1)}$ and the sum @ *k is normalized by* its upper bound – the $IDCG$

Ground Truth

- Positive Result 1 Relevance: 5
- Positive Result 2 Relevance: 3
- Positive Result 3 Relevance: 1

$$\boldsymbol{k = 3}$$

$$DCG@k = \sum_{i=1}^{k} \frac{rel_i}{\gamma_i} = \sum_{i=1}^{k} \frac{rel_i}{\log_2(i+1)}$$

$$= \frac{1}{log_2(1+1)} + 0 + \frac{5}{log_2(3+1)} = 3.5$$

$$IDCG@k = \frac{5}{log_2(1+1)} + \frac{3}{log_2(2+1)} + \frac{1}{log_2(3+1)} = 7.39$$

Ranking Induced by Algorithm

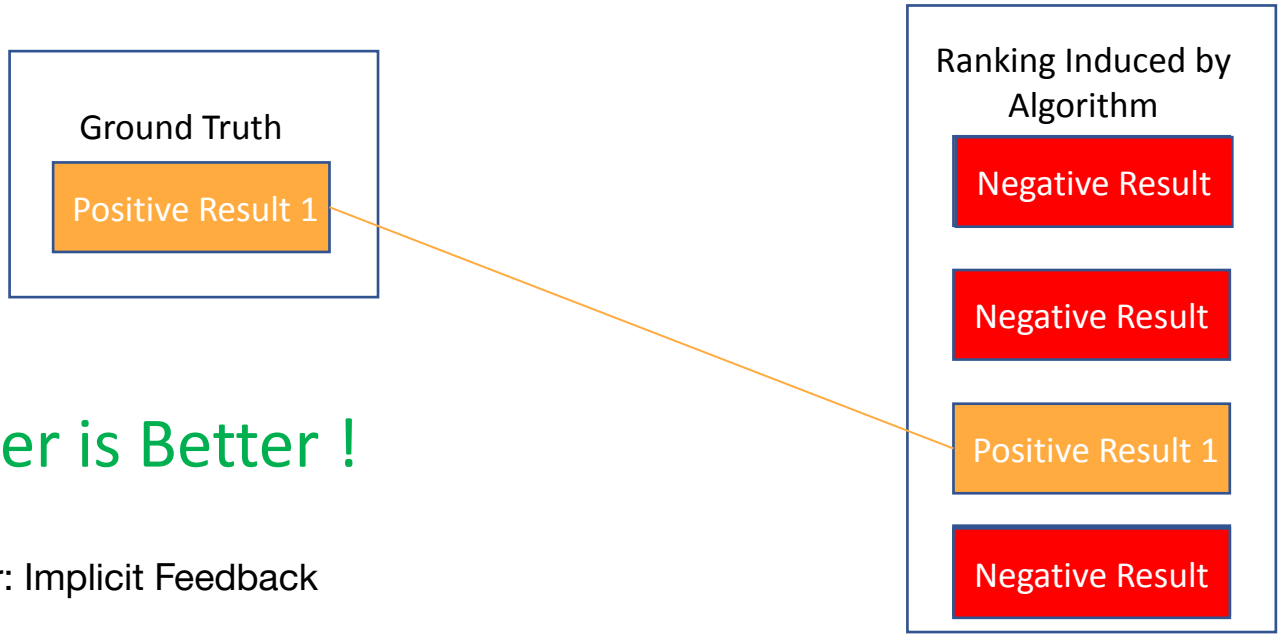
- Positive Result 3
- Negative Result
- Positive Result 1
- Positive Result 2

Used for: Implicit Feedback

$$\boldsymbol{nDCG@k = \frac{DCG}{IDCG} = \frac{3.5}{7.39} = 0.47}$$

MPR - Mean Percentile Rank

Sometimes there is only one “positive” item in the test set…



Lower is Better !

Used for: Implicit Feedback

$$rank_i = 3$$

$$PR = \frac{3}{4} = 0.75$$

Online experiments

Randomized controlled experiments

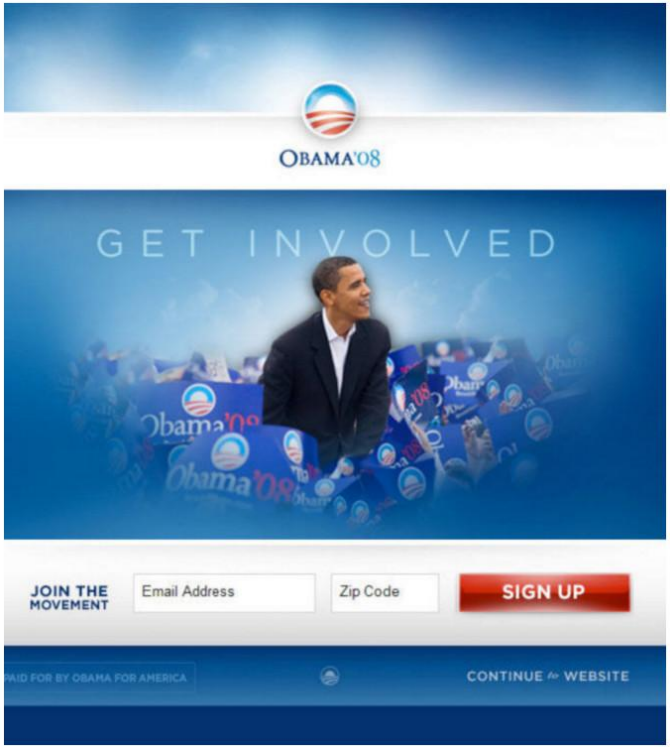
Measure KPIs (Key Performance Indicator) directly

Can compare several variants simultaneously

The ultimate evaluation technique!

Causality: data *vs* the HiPPO

A

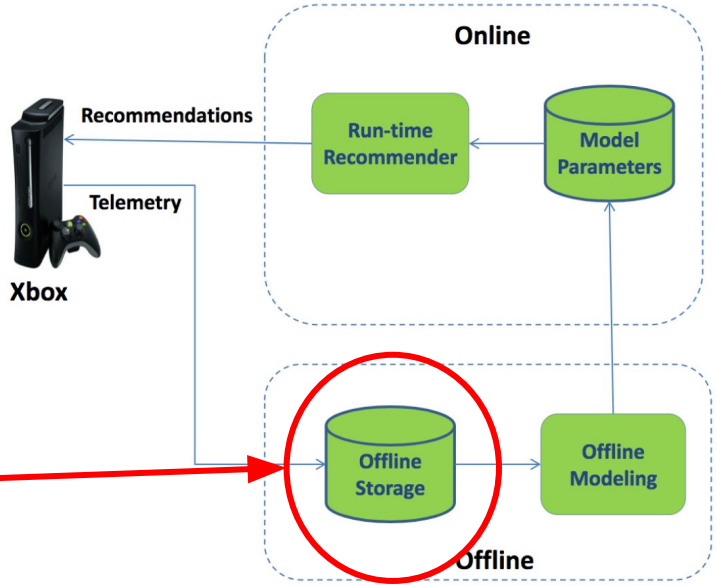


B

Causality, A/B testing!

Does my model cause a change in a KPI (key performance indicator)?

Which changes should we deploy?



We need the right logging here: experiment ID + random user partition (A or B) + feedback...

Resources

A/B testing!

Ron Kohavi's lectures on A/B testing

To watch: KDD 2015 Keynote talk and KDD 2015 Keynote slides PDF

Useful to read

Unexpected Results in Online Controlled Experiments

Seven Pitfalls to Avoid when Running Controlled Experiments on the Web

Practical Guide to Controlled Experiments on the Web: Listen to Your Customers not to the HiPPO

Controlled experiments on the web: survey and practical guide

Class work for Tuesday 13 February

All: Ron Kohavi's lectures on A/B testing

All: To watch: KDD 2015 Keynote talk and KDD 2015 Keynote slides PDF

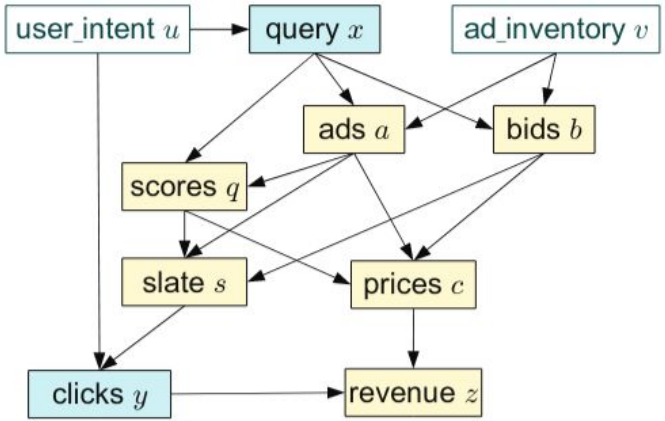
In 4 groups, each present (20 minutes each)

1. Unexpected Results in Online Controlled Experiments
2. Seven Pitfalls to Avoid when Running Controlled Experiments on the Web
3. Practical Guide to Controlled Experiments on the Web: Listen to Your Customers not to the HiPPO
4. Controlled experiments on the web: survey and practical guide

Class work for Tuesday 13 February

Bonus! If you want a step-up, dip into Léon Bottou et al's classic:

Counterfactual Reasoning and Learning Systems: The Example of Computational Advertising



Experimentation caveats

What KPI to measure?

How long to run the experiment?

External factors may influence the results

Cannibalization is hard to account for

Expensive to implement

Can’t compare algorithms before “lighting up”

Practical 7

Building in A/B testing

Practical 7

✅ Wrap either of practicals 1 and 2 in a **system** (a bit of code or API that could serve recommendations), e.g.

```
def createreco(userid, …) -> List[Tuple[int,str]]: ….
```

Here `Tuple[int,str]` says that we return the movie id and title

✅ Add a bit of code to simulate a user giving feedback on the list, and based on that:

✅ Add basic “logging” to your system (i.e. write user id, item id and feedback to disk, for example a CSV file)

Practical 7

✅ Create (say 100) dummy users and simulate new interactions from them (through some rule that you make up)

A/B testing!

✅ Change **one thing** in your model, e.g. down-weighing item biases when making recommendations. Now we have **two versions**, your original and one with the tweak!

Let your API take in a model id so that you could serve a list depending on whether the user is in group A or B:

```
def createreco(userid, versionid) -> List[Tuple[int,str]]: ….
```

✅ Expand your basic “logging” to also write the model id to disk (e.g. user id, item id, feedback, version id)

Practical 7

A/B testing! *(continued)*

✅ **Randomly split** your users into A and B, and do a controlled trial. Simulate them giving feedback. Remember to track the version id: either A or B!

✅ Write each feedback as a line to CSV

✅ Is A or B better?

Show how you measure!

Optimize, optimize, optimize the user experience and other KPIs

using data

lots of data

Final report

📚 Final report 📚

The course contains four practicals, running over three weeks. The practicals will help you derive, code and demonstrate your own prototype of a recommender system using the MovieLens dataset.

You will be evaluated on a final **technical report** that showcases your recommendation engine prototype and all the steps you took to build (and understand) it. It should include detailed equations, derivations, figures, and illustrative and real predictions.

📚 Final report 📚

The final report should be **at least 8 pages long** (without technical appendices). You are welcome to structure it so that it contains “technical appendices” at the end, if there are particularly long derivations that you would like to add.

Your **source code** should accompany your report, and should be:

- saved in Github and
- shared with Github user **upaq**

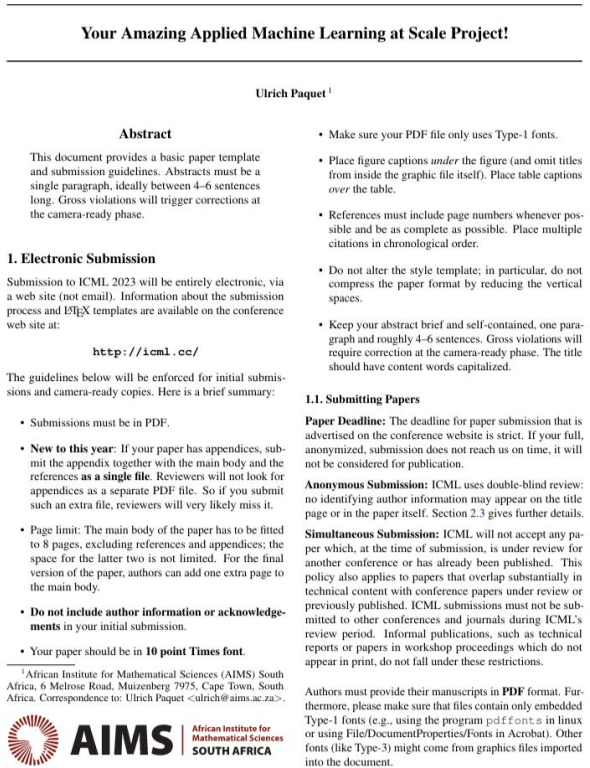
Please also mention your Github source repository link at the end of your “abstract” of the report.

📚 Final report 📚

This is a practice round for wrapping up real research, so…

…your research article should follow the AIMS-ICML Style Guide

We're not really going to submit to ICML 😊; this is the "learning round"!



Your Amazing Applied Machine Learning at Scale Project!

Ulrich Paquet [1]

Abstract

This document provides a basic paper template and submission guidelines. Abstracts must be a single paragraph, ideally between 4–6 sentences long. Gross violations will trigger corrections at the camera-ready phase.

1. Electronic Submission

Submission to ICML 2023 will be entirely electronic, via a web site (not email). Information about the submission process and LaTeX templates are available on the conference web site at:

`http://icml.cc/`

The guidelines below will be enforced for initial submissions and camera-ready copies. Here is a brief summary:

- Submissions must be in PDF.
- **New to this year**: If your paper has appendices, submit the appendix together with the main body and the references **as a single file**. Reviewers will not look for appendices as a separate PDF file. So if you submit such an extra file, reviewers will very likely miss it.
- Page limit: The main body of the paper has to be fitted to 8 pages, excluding references and appendices; the space for the latter two is not limited. For the final version of the paper, authors can add one extra page to the main body.
- **Do not include author information or acknowledgements** in your initial submission.
- Your paper should be in **10 point Times font**.

[1]African Institute for Mathematical Sciences (AIMS) South Africa, 6 Melrose Road, Muizenberg 7975, Cape Town, South Africa. Correspondence to: Ulrich Paquet <ulrich@aims.ac.za>.

AIMS African Institute for Mathematical Sciences SOUTH AFRICA

- Make sure your PDF file only uses Type-1 fonts.
- Place figure captions *under* the figure (and omit titles from inside the graphic file itself). Place table captions *over* the table.
- References must include page numbers whenever possible and be as complete as possible. Place multiple citations in chronological order.
- Do not alter the style template; in particular, do not compress the paper format by reducing the vertical spaces.
- Keep your abstract brief and self-contained, one paragraph and roughly 4–6 sentences. Gross violations will require correction at the camera-ready phase. The title should have content words capitalized.

1.1. Submitting Papers

Paper Deadline: The deadline for paper submission that is advertised on the conference website is strict. If your full, anonymized, submission does not reach us on time, it will not be considered for publication.

Anonymous Submission: ICML uses double-blind review: no identifying author information may appear on the title page or in the paper itself. Section 2.3 gives further details.

Simultaneous Submission: ICML will not accept any paper which, at the time of submission, is under review for another conference or has already been published. This policy also applies to papers that overlap substantially in technical content with conference papers under review or previously published. ICML submissions must not be submitted to other conferences and journals during ICML's review period. Informal publications, such as technical reports or papers in workshop proceedings which do not appear in print, do not fall under these restrictions.

Authors must provide their manuscripts in **PDF** format. Furthermore, please make sure that files contain only embedded Type-1 fonts (e.g., using the program `pdffonts` in linux or using File/DocumentProperties/Fonts in Acrobat). Other fonts (like Type-3) might come from graphics files imported into the document.

1

📚 Final report 📚

Things you should at least cover in your report:

- ✅ Introduction and some past work
- ✅ The problem statement
- ✅ Data set plots (power laws, ratings distributions)
- ✅ A write-up of all your models
- ✅ A long results section!

Only include results from Practicals 1 to 5. (Practicals 6 and 7 are for if you want to explore after this course)

📚 Final report: results 📚

Your results section should contain:

✅ Description of train / test set split

✅ Alternating Least Squares (ALS), biases only. Root Mean Square Error (RMSE) **results** on any MovieLens dataset, train and test

✅ ALS, biases + **U**, **V** matrices (user and movie trait vectors). RMSE **results** on any MovieLens dataset, train and test

📚 Final report: results 📚

✅ Plots of the loss function over training iterations. It should go down monotonically! Or monomotically up if you take the regularized log likelihood 😄

✅ 2D embeddings of the “item trait vectors” $\mathbf{v}_n$, showing that similar movies are embedded together and opposite movies are embedded in “opposite” parts of the space. You’d have to include biases in your training here and use 2D trait vectors in **U** and **V**, and then plot the 2D vectors in **V** for a selection of movies

📚 Final report: “model” section and results 📚

✅ Use the MovieLens genres metadata

movies.csv

```
movieId,title,genres
1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy
2,Jumanji (1995),Adventure|Children|Fantasy
3,Grumpier Old Men (1995),Comedy|Romance
```

as part of your model. Describe your altered loss function in sufficient detail, including the optimization steps in the “modeling” section.

✅ For a model with two-dimensional trait vectors (2D latent dimensions), plot the learned embedding of the genre trait vectors, and reason how it makes sense.

📚 Final report 📚

✅ Describe and show how many ALS training iterations you used. Why?

✅ Describe and show how hyperparameters were chosen, e.g. λ and τ

✅ Describe and show how you chose the “best” number of latent dimensions for your model, and at which point you achieved diminishing returns

Hint: You can try K = 0, 2, 4, 8, 16 and possibly 32. Don’t bother with K = 50 or 100 or other huge numbers; you’ll just spend most of your time waiting for your algorithm to converge.

📚 Final report: results 📚

✅ Can you show results on **all 25M (or 32M) ratings**, including implementation to make your algorithm run faster than the most naive implementation

✅ Show that if a user liked movie X (your choice) the recommendations Y make sense

For instance, create a “dummy user” that gave some movie like a “Lord of the Rings” five stars. Are the top recommendations other “Lord of the Rings” movies? You’ll need the 25M dataset for this to start looking really nice.

🍒 Yes, this is the cherry on the cake! 🎂 You have to show this if you want to “impress”! See the last slide of Practical 4

📚 Final report: results 📚

Usually, in modern machine learning papers, there is a link to **publicly accessible code** so that anyone could reproduce your work.

Please include:

✅ A Github reference to legible ALS (and other) code that you wrote yourself

✅ Nice code in Github (e.g. PEP 8 for Python)

📚 Final report 📚

The marks breakdown of the final report is as follows:

✅ 5% Data set plots (power laws, ratings distributions)

✅ 5% Alternating Least Squares (ALS), biases only. Root Mean Square Error (RMSE) **results** on any MovieLens dataset

✅ 2% If there are additional results on a train / test set split that you devised

✅ 10% ALS, biases + **U**, **V** matrices (user and movie trait vectors). RMSE **results** on any MovieLens dataset

✅ 2% If there are additional results on a train / test set split

In real life you might use a train / validation / test split of your data to ensure that you don't accidentally overfit on your test set

📚 Final report 📚

✅ 3% Plots of the negative loss function over training iterations. It should go up monotonically! Or monomotically down if you don't take minus 😄

✅ 11% Legible ALS (and other) code that you wrote yourself

✅ 2% Coding style (e.g. PEP 8 for Python)

✅ 5% 2D embeddings of the "item trait vectors" $\mathbf{v}_n$, showing that similar movies are embedded together and opposite movies are embedded in "opposite" parts of the space. You'd have to include biases in your training here and use 2D trait vectors in **U** and **V**, and then plot the 2D vectors in **V** for a selection of movies

📚 Final report 📚

✅ 4% Use the MovieLens genres metadata

movies.csv

```
movieId,title,genres
1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy
2,Jumanji (1995),Adventure|Children|Fantasy
3,Grumpier Old Men (1995),Comedy|Romance
```

as part of your model. Describe your altered loss function in sufficient detail, including the optimization steps.

For a model with two-dimensional trait vectors (2D latent dimensions), plot the learned embedding of the genre trait vectors, and reason how it makes sense.

📚 Final report 📚

✅ 12% Bonus if your results are on **all 32M ratings**, including implementation to make your algorithm run faster than the most naive implementation

✅ 5% Show that if a user liked movie X (your choice) the recommendations Y make sense

For instance, create a “dummy user” that gave some movie like a “Lord of the Rings” five stars. Are the top recommendations other “Lord of the Rings” movies? You’ll need the 25M dataset for this to start looking really nice.

✅ 3% What are the most polarizing movies? And the least polarizing movies? Did you make your claim without overfitting being an explanation?

Some polarizing that quickly narrows down a user’s taste once you know whether they like / dislike it. Typically movies with unusually long trait vectors **v**

📚 Final report 📚

✅ 3% Describe and show how hyperparameters were chosen, e.g. λ and τ

✅ 3% Describe and show how you chose the “best” number of latent dimensions for your model, and at which point you achieved diminishing returns

📚 Final report 📚

✅ ~~7%~~ ~~Show a practical algorithm at work on **implicit feedback data**, evaluated with Precision@k and Recall@k for different values of k (say 10, 20, 100).~~

~~You could take all 4+ ratings on MovieLens 25M as the implicit positive signal, and ignore the rest. Code Bayesian Personalized Ranking (BPR) to illustrate a ranking-based loss function.~~

~~Your results *might not look that good* due to your creation of data – your model is not trained on real implicit feedback data – but you can show that you understand how implicit feedback data is modeled.~~

✅ ~~5%~~ ~~Show, though code of a mocked up system, how you would implement an **A/B test** on your users.~~

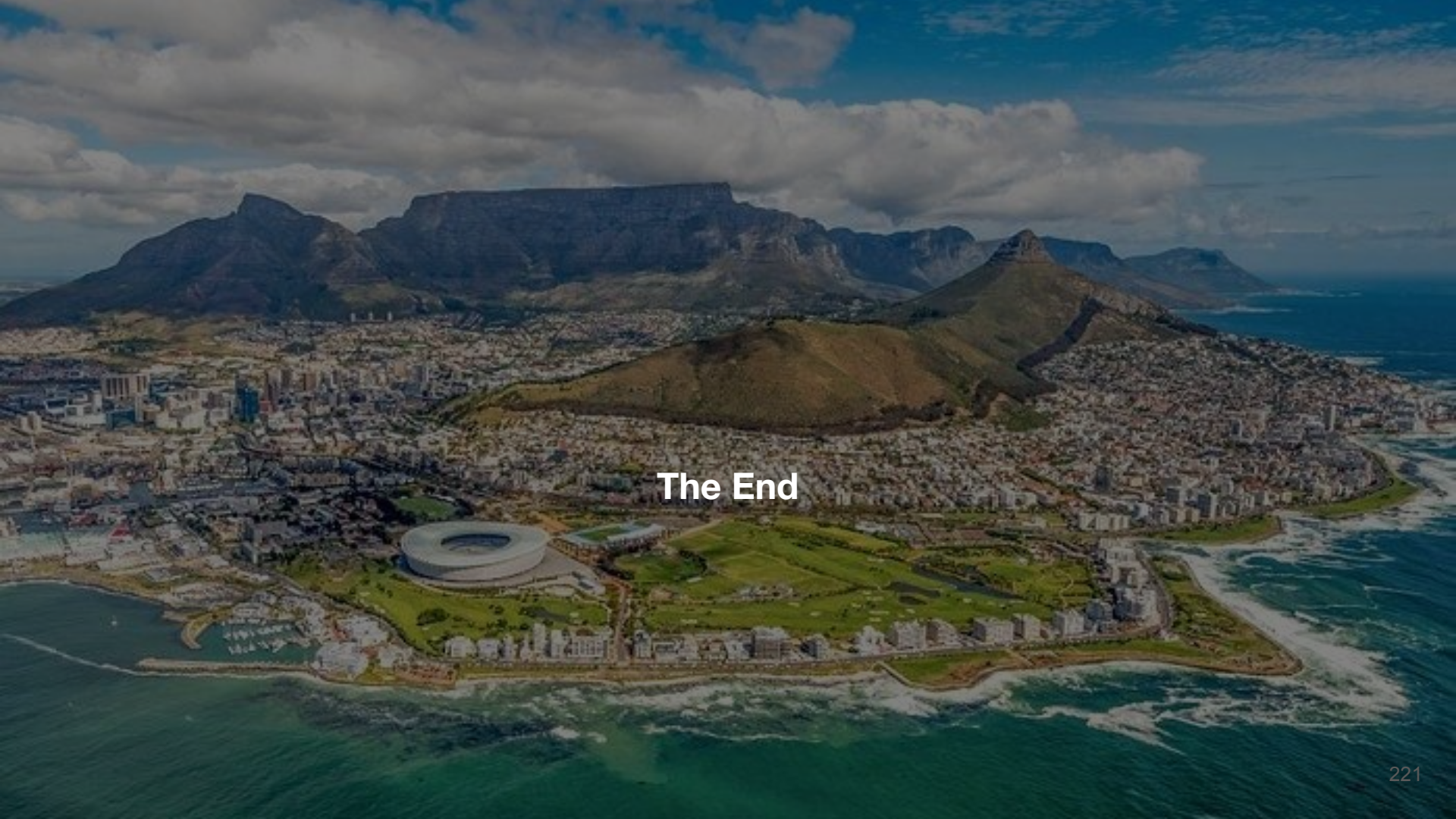
✅ ~~3%~~ ~~Show a simulated A/B test with any algorithmic change, including the t-test to see if your change makes a statistically significant difference~~

📚 Final report 📚

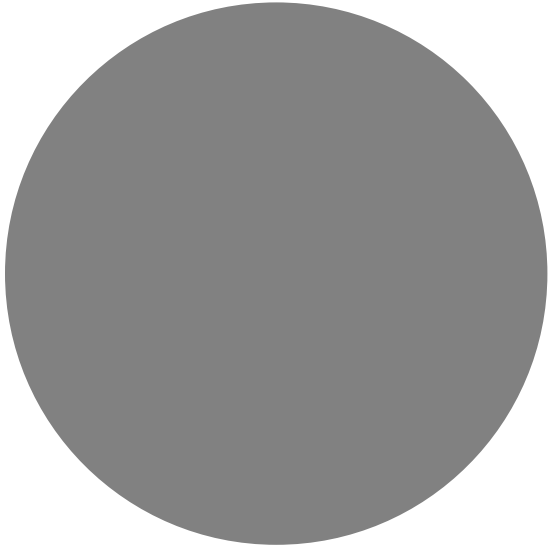
✅ 10% Writing, writing, writing 😄. A clean layout with an introduction, logical sections and subsections, neatly typeset equations with correct sub-indexing, correctly numbered figures that are referenced in the text, etc. LaTeX is your friend.

💯 **100%** The brings you to 100% (renormalized). You can go above and beyond and squeeze in bonus marks 😄:

✅ 7% **Optional** self-work bonus: Go beyond a “single point” estimate when inferring the parameters of **U**, **V** and the biases **b**, and implement a Variational Bayes solution. Present your derivations, code and results


The End

Evaluation of Recommender Systems



Train, Validation and Test Datasets

- Train Dataset – The sample of data used to fit the model

  This is the actual data we use to train the model and learn parameters

- Validation Dataset - The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters.

  Models often have many hyperparameters (e.g., dimensionality, number of layers, architecture choices, optimization choices, etc.). The validation set is used only after the model is trained on the training dataset. It is used to evaluate the quality of the model under different hyper-parameter choices. Often the test dataset is used for tuning hyperparameters. However, as the model becomes biased towards this dataset, it is impossible to determine the actual quality of the model without a separate set.

- Test Datasets - The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.

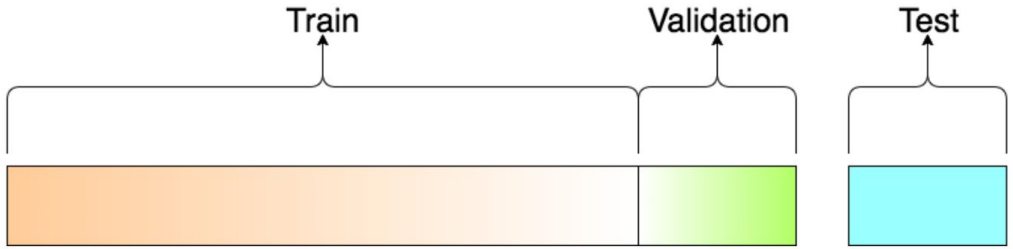
  The test set is used once at the very end to evaluate the actual performance of a recommender systems.

Definitions taken from Jason Brownlee's article: **What is the Difference Between Test and Validation Datasets?**

Train, Validation and Datasets Test Cont.

- Most machine learning datasets are created by randomly splitting the data into train, validation and test.
- Cross validation can be achieved by repeating the this split multiple times.
- Recommendations datasets are time-dependent
  - The split usually follows the actual timeline of events
  - Random split usually yield an “easier” datasets that are not “realistic”

Towards Data Science
Sharing concepts, ideas, and codes.



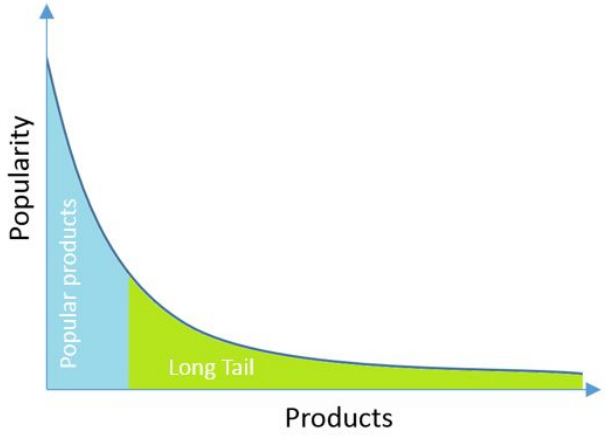
A visualisation of the splits

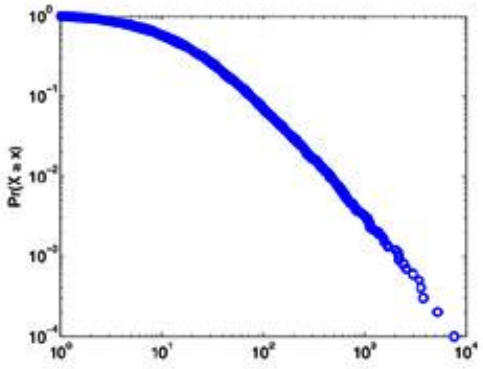
Power-law Distributions in Recommendation Datasets

- Recommendation datasets often exhibit a power-law distribution.
- In power-law distributions, the probability for an item to have a certain number of ratings x is decreasing according to:

$$Pr(x) \propto ax^{-b}$$

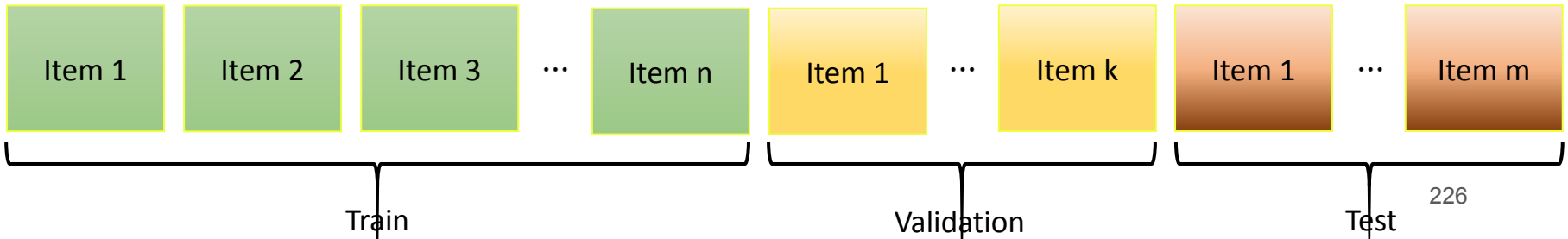
- Power-law is a very skewed distribution in which a small number of popular items dominate the datasets and most items have a very limited number o ratings.
- Power-law distributions in practice can be identified by plotting popularity vs. number of ratings on a log-log plot.
- The number of ratings per user often follows a “long-tail” skewed distribution as well.





Dataset Partition - Coping with Long Tail Distributions

- Recommendation datasets often exhibit a power-law distribution of ratings per items and per users.
- We do not want the recommender system to be biased towards “heavy” users. Instead, we want the recommender system to “care” about all the users the same.
- Removing ratings from heavy users or items is equivalent to deleting potentially useful data and changing user-item relations.
- In many cases (e.g., Netflix, KDD Cup 2011), the last *m* ratings of each users are left for the validation set and the last *n* ratings are left for the test set.



What to do with features? (also called side-information)

Various options…

Dataset examples

movies.csv

```
movieId,title,genres
1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy
2,Jumanji (1995),Adventure|Children|Fantasy
3,Grumpier Old Men (1995),Comedy|Romance
```

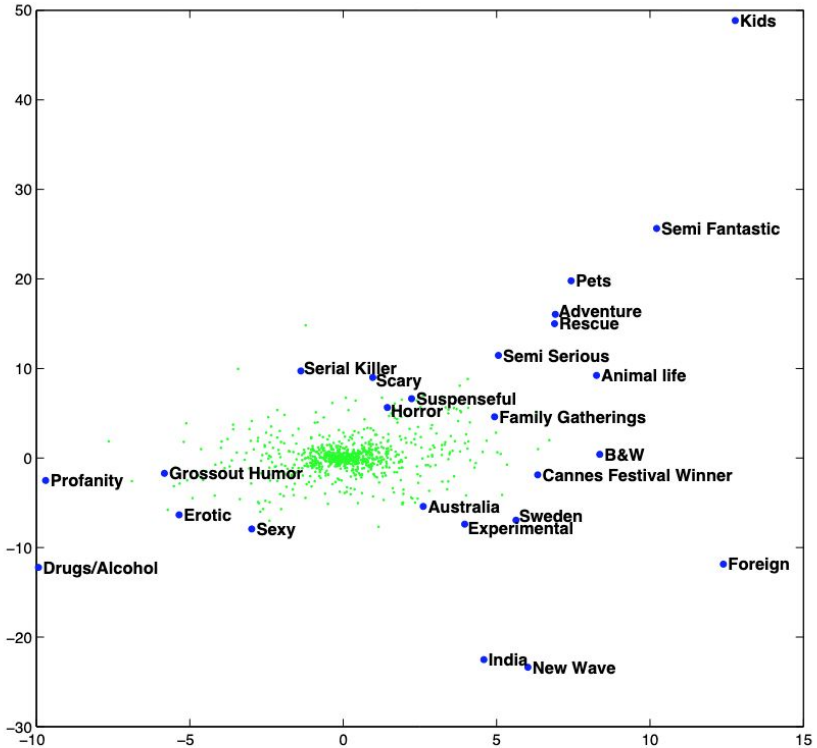


Figure 6: A visualization of the mean feature-vectors in the Xbox Movies dataset trained with a $D = 2$ dimensional *MF-EFS* model. The sparsity is evident by the concentration of near zero feature vector norms. We added text labels to the smaller number of "informative" features – those with a high norm.

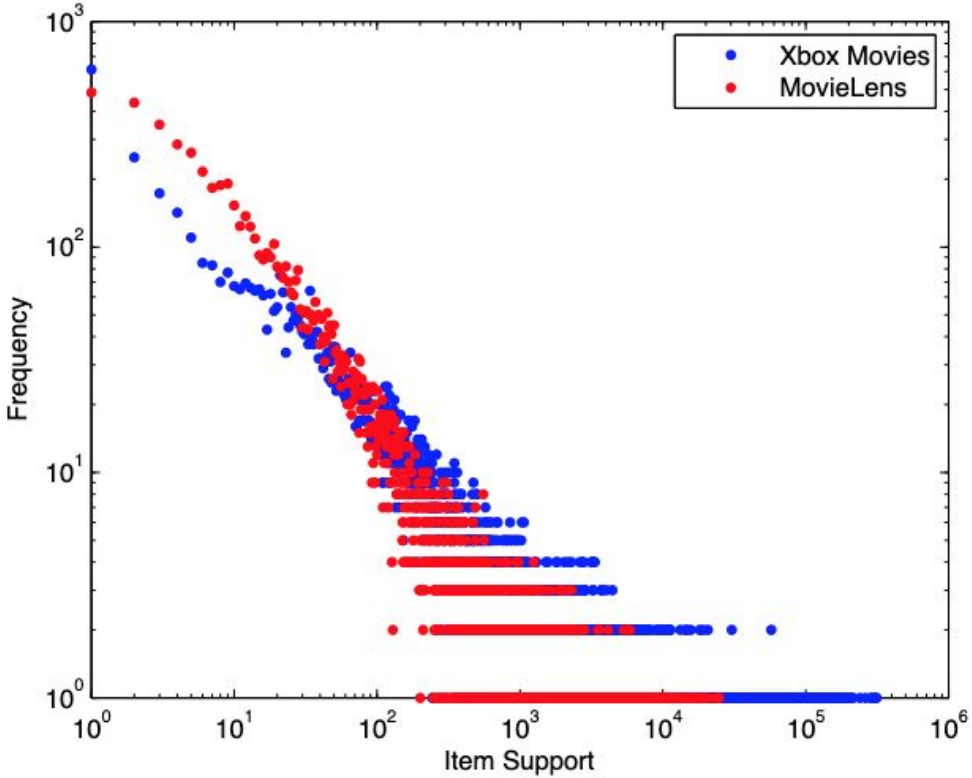
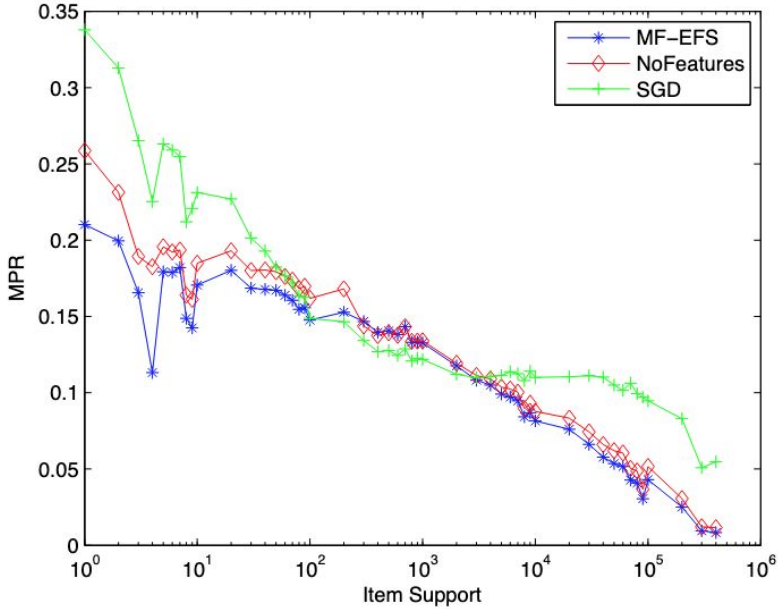
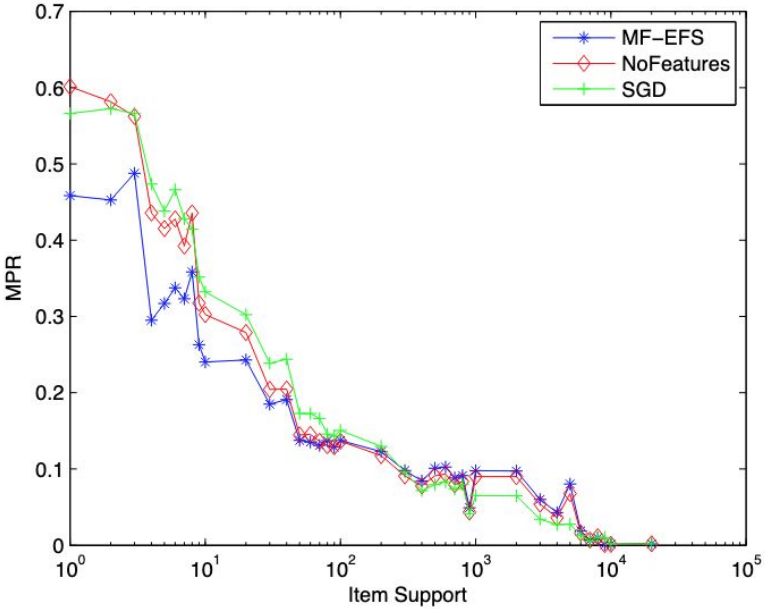


Figure 4: Item support distribution of Xbox Movies and MovieLens datasets. The Xbox Movies dataset is larger and its distribution is more skewed.



(a) Xbox Movies



(b) MovieLens

Figure 3: Mean Percentile Rank (MPR) vs. Item Support (lower is better). The contribution of the features in *MF-EFS* is clearly evident in improving cold-items recommendations.


I MADE THIS
LAUNCH TEAM 2013

Reservoir sampling

Weighted reservoir sampling

$$p(\mathbf{f}_k|\tau_k) = \mathcal{N}(\mathbf{f}_k\,;\,\mathbf{0},\,\tau_k^{-1}\mathbf{I}) \quad \text{and} \quad p(\tau_k|\alpha,\beta) = \mathcal{G}(\tau_k\,;\,\alpha,\,\beta)$$

$$p(\mathbf{f}_k|\alpha,\,\beta) = \int p(\mathbf{f}_k|\tau_k)\,p(\tau_k|\alpha,\,\beta)\,\mathrm{d}\tau_k$$

$$= \frac{\Gamma\left(\frac{\nu+D}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)\,(\nu\pi)^{\frac{D}{2}}\,|\boldsymbol{\Sigma}|^{\frac{1}{2}}}\left[1+\frac{\alpha}{\beta\nu}\mathbf{f}_k^\top\mathbf{f}_k\right]^{-\frac{\nu+D}{2}}$$

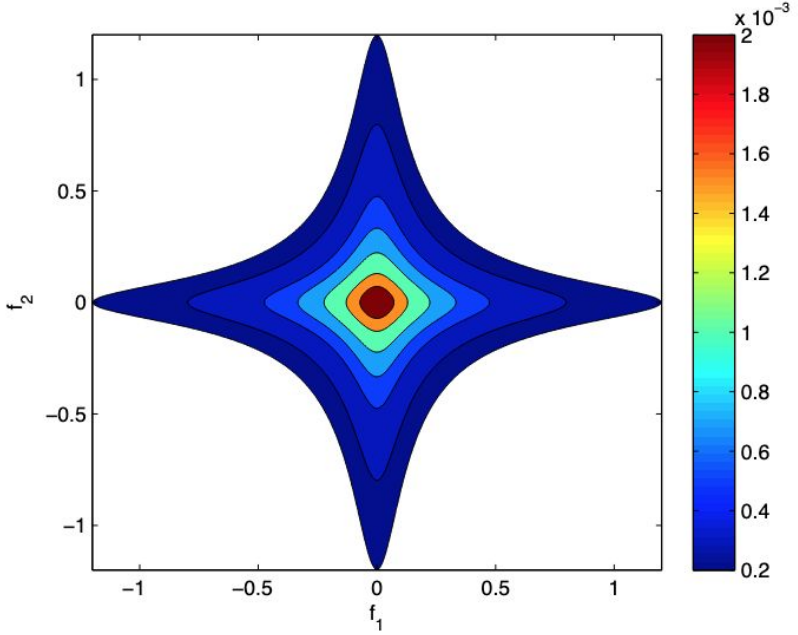
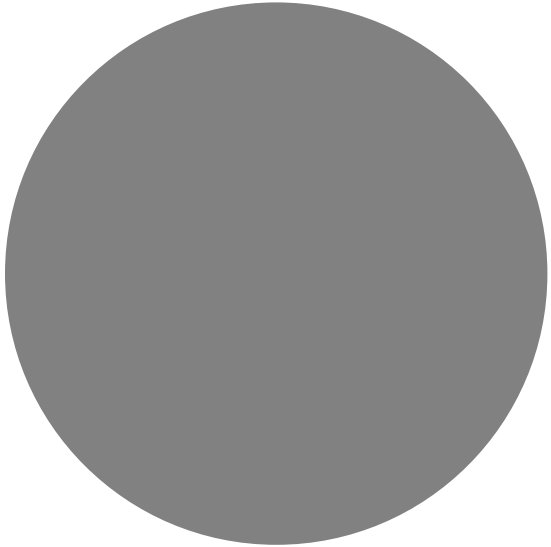


Figure 2: Contours of the probability mass of the *effective* prior for two feature vectors (f_1 and f_2): $p(f_1, f_2|\alpha = 0.01, \beta = 0.01)$. For the sake of the visualization, the feature vectors here are one-dimensional ($D = 1$). The heavy tails originating from the underlaying t-distributions are clearly seen along the axes. As the number of features is higher, this effect results in a concentration of probability mass along the corners which encourages sparse solutions.

$$p(\mathbf{y}_n|\{\mathbf{f}_k\}, L_n, \tau_y) = \mathcal{N}\left(\mathbf{y}_n\,;\, \frac{1}{\sqrt{|L_n|}}\sum_{k\in L_n}\mathbf{f}_k,\, \tau_y^{-1}\mathbf{I}\right)$$

Evaluation Techniques



Offline Evaluation Techniques

Issues with Offline Evaluation Techniques

Online Evaluation

Off-policy Evaluation

$RMSE$- Root Mean Square Error

RMSE is computed by averaging the squared errors over all user item pairs,$(u, i) \in \mathcal{R}$

$$RMSE = \sqrt{\frac{1}{|\mathcal{R}|} \sum_{(u,i)\in\mathcal{R}} (\hat{r}_{ui} - r_{ui})^2}$$

Used for: Explicit Ratings

Issues with RMSE:

- Tend to emphasize large errors rather than preserving fine neighborhoods. However, the recommendations are chosen only from the very top
- Tend to emphasize popular items that everyone rates

$wRMSE$- Weighted Root Mean Square Error

This variant of RMSE is achieved by assigning each data point a weight,w_{ui}, based on its importance.

$$RMSE = \sqrt{\frac{1}{\sum w_{ui}} \sum_{(u,i)\in\mathcal{R}} w_{ui} \cdot (\hat{r}_{ui} - r_{ui})^2}$$

Possible choices of the weights:

- Lower weights for popular items
- Higher rates when more confidence is available

Used for: Explicit Ratings

MAE - Mean Absolute Error

MAE is computed by averaging the absolute errors over all user item pairs,$(u, i) \in \mathcal{R}$

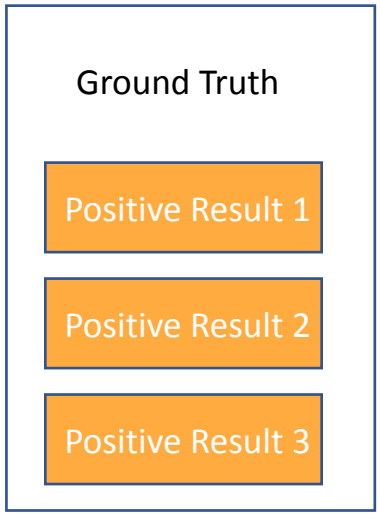
$$MEA = \frac{1}{|\mathcal{R}|} \sum_{(u,i)\in\mathcal{R}} |\hat{r}_{ui} - r_{ui}|$$

Unlike RMSE, MAE does not emphasize large errors.

Less common than RMSE due to the difficulties working with its derivative (+1,-1) .

Used for: Explicit Ratings

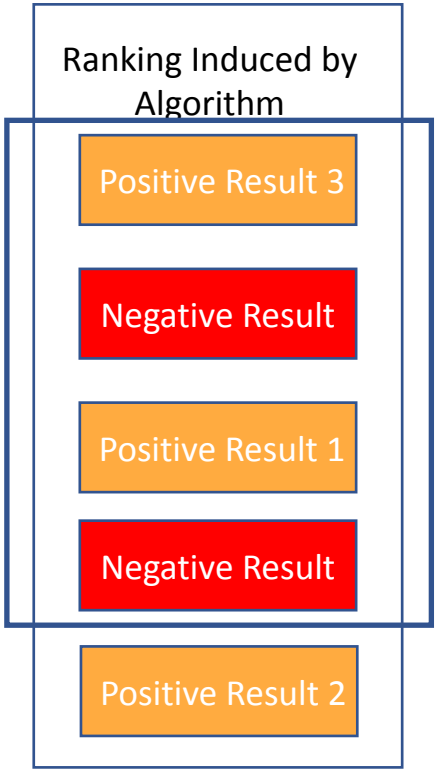
Precision@k / Recall@k



$$\boldsymbol{k = 4}$$

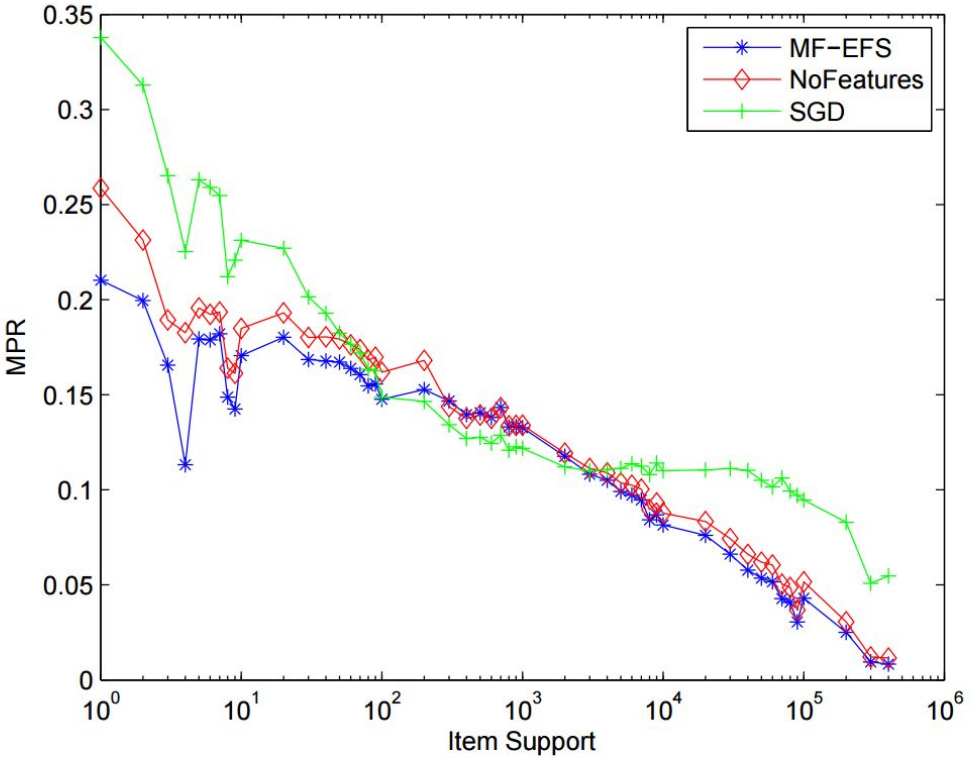
$$\boldsymbol{precision@k} = \frac{\#tp}{\#tp + \#fp} = \frac{2}{2+2} = \frac{1}{2}$$

$$\boldsymbol{recall@k} = \frac{\#tp}{\#tp + \#fn} = \frac{2}{2+1} = \frac{2}{3}$$



Used for: Implicit Ratings

MPR in Xbox

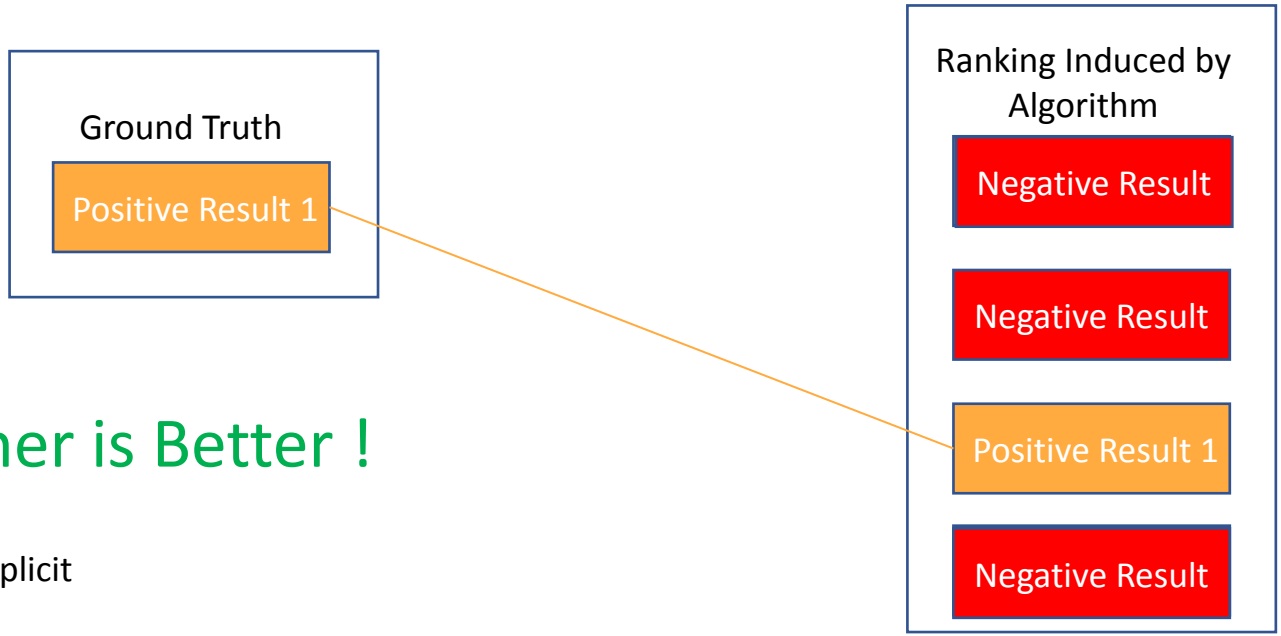


(a) Xbox Movies

Xbox Movies Recommendations: Variational Bayes Matrix Factorization with Embedded Feature Selection by Noam Koenigstein and Ulrich Paquet

MRR- Mean Reciprocal Rank

Sometimes there is only one “positive” item in the test set…



Higher is Better !

$$rank_i = 3$$
$$RR = \frac{1}{3}$$

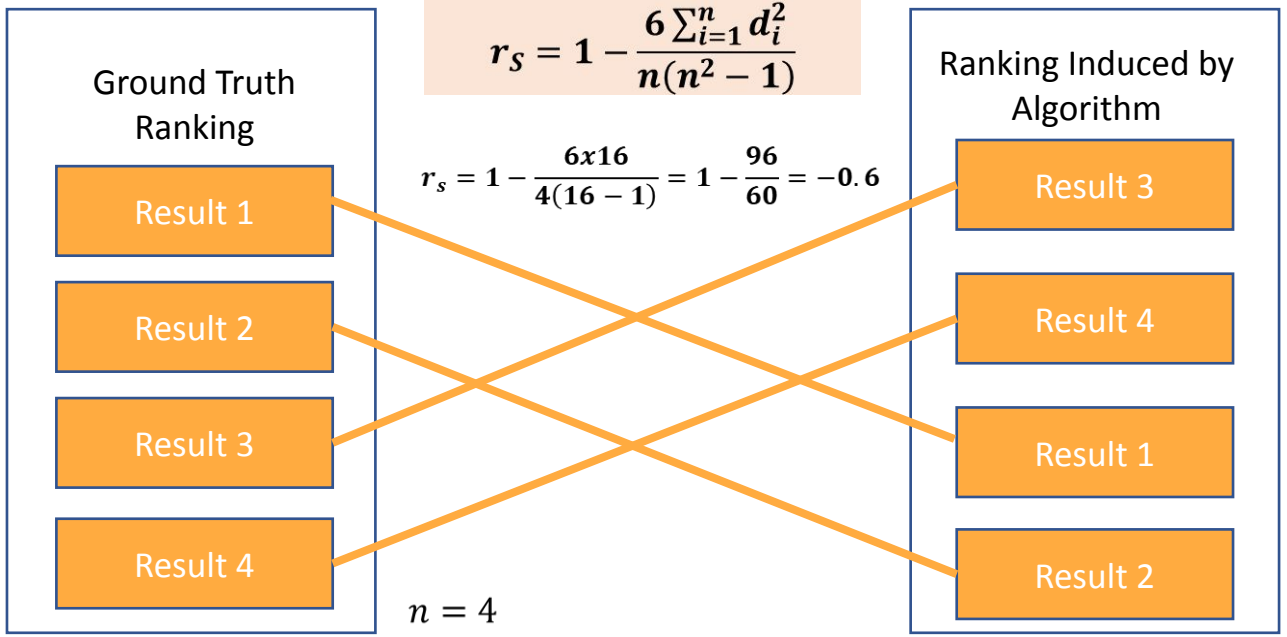
Used for: Implicit Ratings

Spearman's Rank Correlation Coefficient

In scenarios where we want to emphasize the full ranking, we may compare the ranking of the algorithm to a reference ranking.

The Spearman correlation coefficient is defined as the Pearson correlation coefficient between the rank variables.

$$r_s = 1 - \frac{6\sum_{i=1}^{n} d_i^2}{n(n^2-1)}$$



$$r_s = 1 - \frac{6x16}{4(16-1)} = 1 - \frac{96}{60} = -0.6$$

$n = 4$

$d_1 = r_1 - \hat{r}_1 = 1 - 3 = -2$

$d_2 = r_2 - \hat{r}_2 = 2 - 4 = -2$

$d_3 = r_3 - \hat{r}_3 = 3 - 1 = 2$

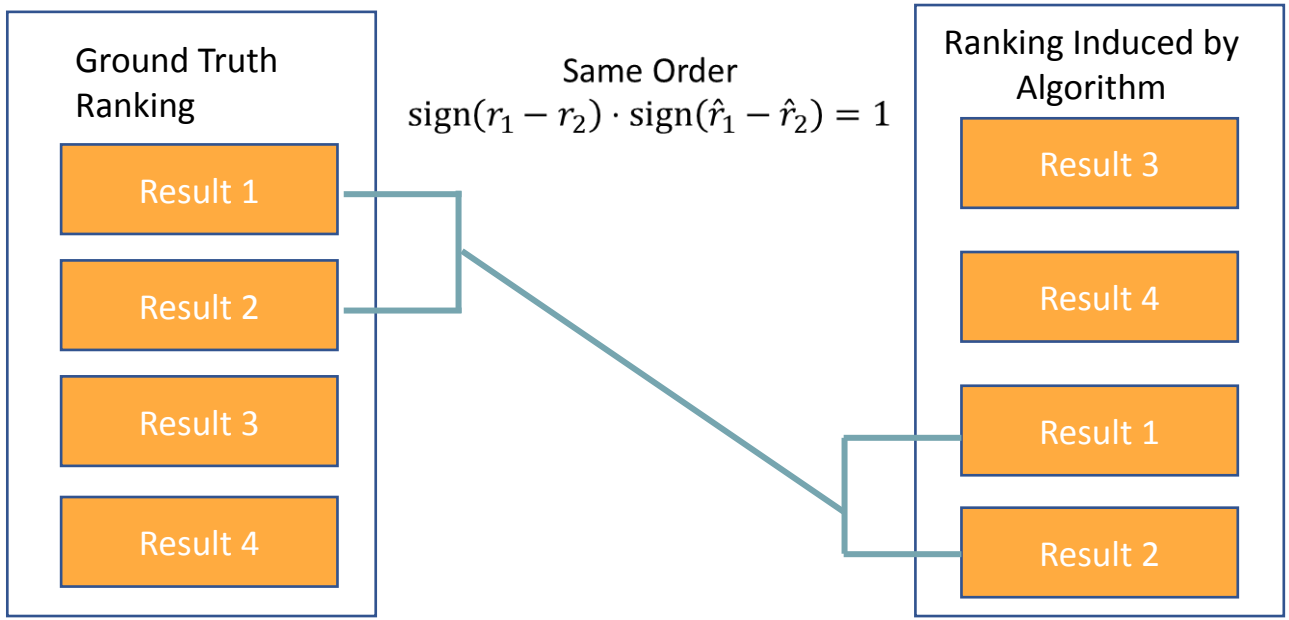
$d_4 = r_4 - \hat{r}_4 = 4 - 2 = 2$

$\sum_{i=1}^{n} d_i^2 = 16$

Kendall's Rank Correlation Coefficient

In scenarios where we want to emphasize the full ranking we may compare the ranking of the algorithm to a reference ran

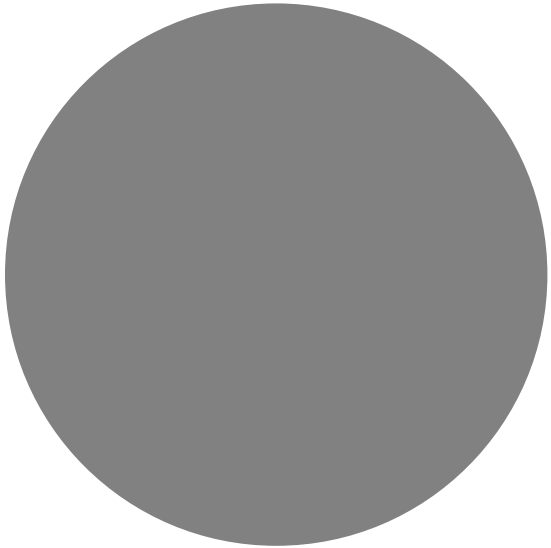
$$\tau = \frac{(\#\textbf{concardant pairs}) - (\#\textbf{discordant pairs})}{\binom{n}{2}}$$
$$= \frac{2}{n(n-1)} \sum_{i<j} \textbf{sign}(r_i - r_j) \cdot \textbf{sign}(\hat{r}_i - \hat{r}_i)$$



Offline Techniques – Open Questions

- How do we measure the importance/ relevance of the positive items?
  - Long tail items are more important. But how do we quantify?
  - How many items do we care to recommend?
- Should the best item be the first item?
  - Maybe the best item should be in the middle?
- What about diversity?
- What about contextual effects?
- What about items fatigue?

Classical Categorization of Recommender Systems



Collaborative Filtering vs Content Based

Types of CF Algorithms

CF Examples

CB Examples

Classical Categorization of Recommender Systems

- Collaborative Filtering (CF) Methods:
  - Recommend Items Liked by Similar Users
  - Doesn’t use items meta-data, attributes, explicit features or “content”
  - Favors popular content
  - Suffers from the items/ users “cold-start” problem

- Content Based (CB) Methods:
  - Uses attributes of users / items
  - Recommends items that are similar to those liked/purchased by the user in the past
  - Generally, considered less accurate than CF methods
  - Requires items attributes (content) but does not suffer from “cold-start”

- Hybrid Models:
  - Utilize both CF information as well as CB information for better recommendations
  - Many Deep Learning models are dealing with “multi-views”
  - Usually, the contribution of the CB information is negligible in the presence of CF inforamtion

Types of Collaborative Filtering Algorithms

- **Memory Based CF:**
  - The memory-based approach uses user rating data to compute the similarity between users or items. Typical examples of this approach are neighbourhood-based CF.

- **Model Based CF:**
  - In this approach, models are developed using different data mining, machine learning algorithms to predict users' rating of unrated items. The most prominent approach is Matrix Factorization (MF).

- **Deep Learning Methods:**
  - In recent years a number of neural and deep-learning techniques have been proposed. Some generalize traditional Matrix factorization algorithms via a non-linear neural architecture, or leverage new model types like Variational Autoencoders.

  - While deep learning has been applied to many different scenarios: context-aware, sequence-aware, social tagging etc. its real effectiveness when used in a simple collaborative recommendation scenario has been put into question. A systematic analysis of publications applying deep learning or neural methods to the top-k recommendation problem, published in top conferences (SIGIR, KDD, WWW, RecSys), has shown that on average less than 40% of articles are reproducible.

    Ferrari Dacrema, Maurizio; Cremonesi, Paolo; Jannach, Dietmar (2019). "Are We Really Making Much Progress? A Worrying Analysis of Recent Neural Recommendation Approaches". Proceedings of the 13th ACM Conference on Recommender Systems. ACM: 101–109.

CF Examples – Neighborhood Models

- Item Based Neighborhood Model:
  - Compute item-to-item similarities matrix
  - For each user, find the most similar items to the items she liked
- User Based Neighborhood Model:
  - Compute user-to-user similarity matrix
  - For each user, find her most similar users and recommend items that these users liked, but she did not consumed yet.

Pointwise Similarity Functions

- Computing users or items similarities

  - **Euclidean Distance**:

$$EucDist(x_i, x_j) = \sqrt{\sum_{n=1}^{N} \left(x_i(n) - x_j(n)\right)^2}$$

  - **Cosine Similarity**:

$$CosSim(x_i, x_j) = \frac{x_i^T x_j}{\|x_i\| \|x_j\|} = \frac{\sum_{n=1}^{N} x_i(n) \cdot x_j(n)}{\sqrt{\sum_{n=1}^{N} x_i(n)^2} \sqrt{\sum_{n=1}^{N} x_j(n)^2}}$$

  - **Pearson Correlation**:

$$Corr(x_i, x_j) = \frac{\sum_{n=1}^{N} (x_i(n) - \bar{x_i}) \left(x_j(n) - \bar{x_j}\right)}{\sqrt{\sum_{n=1}^{N} (x_i(n) - \bar{x_i})^2} \sqrt{\sum_{n=1}^{N} \left(x_j(n) - \bar{x_j}\right)^2}} = CosSim = \left(x_i - \bar{x_i}, x_j - \bar{x_j}\right)$$