INFO 2950: Intro to Data Science

Lecture 22
2023-11-13

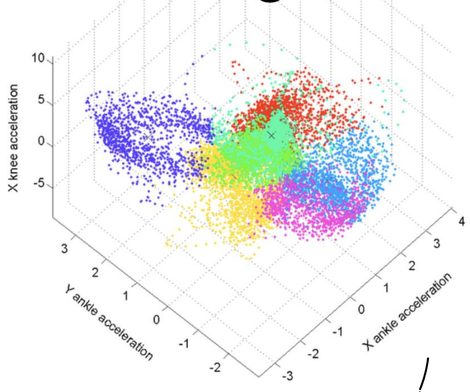
Agenda

1. High-dimensional data for recommendations
    a. Similarities

2. Matrix multiplication
    a. Sparse storage formats
    b. Matrix factorization (SVD)
    c. Netflix Prize

High dimensional data

3D clustering plot: different human runners' gaits



https://www.researchgate.net/figure/3D-plot-of-clusters-obtained-from-the-k-means-algorithm_fig5_278668507

High dimensional data

How do we use high dimensional data to make **personalized recommendations?**

Netflix: millions of users, thousands of videos, lots of ratings

What movie would you suggest?

For someone who likes each of the following, what movie do you suggest?

- The Godfather ____________
- Spirited Away ____________

What movie would you suggest?

For someone who likes each of the following, what movie do you suggest?

- The Godfather The Godfather Part II, ...
- Spirited Away Princess Mononoke, ...

Movie rating data

We have data for lots of "Netflix" users

	User 1	User 2	User 3	User 4	...	User 13435
Airplane!	9	6		7		
Akira		4	7	8		8
Aladdin	6			7		
Alexander Nevsky				6		
...						
Zoolander			9	5		7

Movie rating data

We have data on the different movies in the "Netflix" catalog

	User 1	User 2	User 3	User 4	...	User 13435
Airplane!	9	6		7		
Akira		4	7	8		8
Aladdin	6			7		
Alexander Nevsky				6		
...						
Zoolander			9	5		7

Movie rating data

For each movie that the user has seen and rated out of 10, we store the rating*

	User 1	User 2	User 3	User 4	...	User 13435
Airplane!	9	6		7		
Akira		4	7	8		8
Aladdin	6			7		
Alexander Nevsky				6		
...						
Zoolander			9	5		7

Movie rating data

For each movie that the user has seen and rated out of 10, we store the rating*

*in real life, the relevant metrics might be minutes of movie watched, # times watched, etc.

	User 1	User 2	User 3	User 4	...	User 13435
Airplane!	9	6		7		
Akira		4	7	8		8
Aladdin	6			7		
Alexander Nevsky				6		
...						
Zoolander			9	5		7

Movie rating data

Sparse data!

Most users are only watching a tiny fraction of the catalog!

	User 1	User 2	User 3	User 4	...	User 13435
Airplane!	9	6		7		
Akira		4	7	8		8
Aladdin	6			7		
Alexander Nevsky				6		
...						
Zoolander			9	5		7

Movie rating data

How do we use this dataset to recommend a movie to a specific user?

	User 1	User 2	User 3	User 4	...	User 13435
Airplane!	9	6		7		
Akira		4	7	8		8
Aladdin	6			7		
Alexander Nevsky				6		
...						
Zoolander			9	5		7

One way: find other people who liked similar movies

	User 1	User 2	User 3	User 4	...	User 13435
Airplane!	9	6		7		
Akira		4	7	8		8
Aladdin	6			7		
Alexander Nevsky				6		
...						
Zoolander			9	5		7

One way: find other people who liked similar movies

Step 1. Find a set of users whose ratings are "similar" to User 13435

	User 1	User 2	User 3	User 4	...	User 13435
Airplane!	9	6		7		
Akira		4	7	8		8
Aladdin	6			7		
Alexander Nevsky				6		
...			5			
Zoolander			9	5		7

One way: find other people who liked similar movies

Which User might be most similar to User 13435?

	User 1	User 2	User 3	User 4	...	User 13435
Airplane!	9	6		7		
Akira		4	7	8		8
Aladdin	6			7		
Alexander Nevsky				6		
...			5			
Zoolander			9	5		7

One way: find other people who liked similar movies

Which User might be most similar to User 13435?

User 3 has also watched both Akira and Zoolander, and rated them comparably!

	User 1	User 2	User 3	User 4	...	User 13435
Airplane!	9	6		7		
Akira		4	7	8		8
Aladdin	6			7		
Alexander Nevsky				6		
...			5			
Zoolander			9	5		7

One way: find other people who liked similar movies

But, how could we tell this is the case using only the data, and not our own intuition?

	User 1	User 2	User 3	User 4	...	User 13435
Airplane!	9	6		7		
Akira		4	7	8		8
Aladdin	6			7		
Alexander Nevsky				6		
...			5			
Zoolander			9	5		7

One way: find other people who liked similar movies

Step 1. Find a set of users whose ratings are “similar” to User 13435

Similarity can be calculated with metrics like Pearson correlation, or “cosine similarity” = 1 - “cosine distance”

	User 1	User 2	User 3	User 4	…	User 13435
Airplane!	9	6		7		
Akira		4	7	8		8
Aladdin	6			7		
Alexander Nevsky				6		
…			5			
Zoolander			9	5		7

One way: find other people who liked similar movies

Each column is a vector:
User 1 =
(9,0,6,0,...,0)
User 3 =
(0,7,0,0,5,...,9)
User 13435 =
(0,8,0,0,0,...,7)

	User 1	User 2	User 3	User 4	...	User 13435
Airplane!	9	6		7		
Akira		4	7	8		8
Aladdin	6			7		
Alexander Nevsky				6		
...			5			
Zoolander			9	5		7

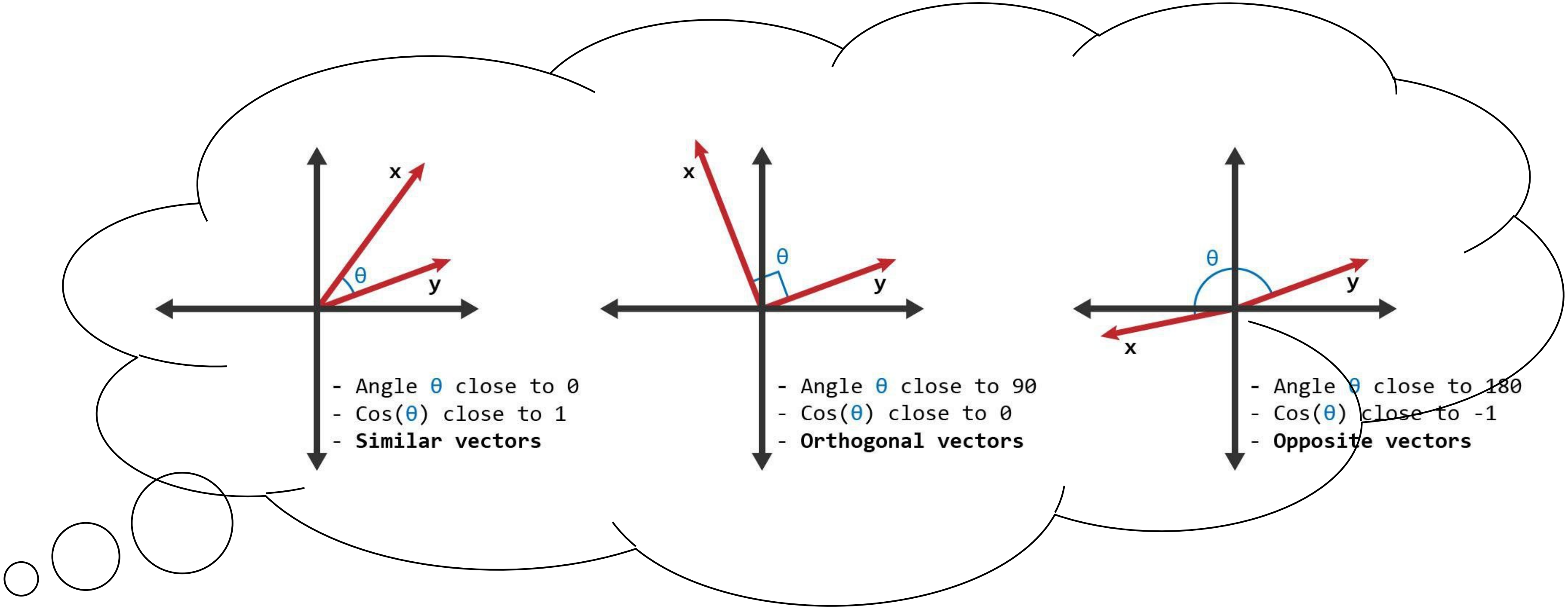
One way: find other people who liked similar movies

Each column is a vector:
User 1 =
(9,0,6,0,...,0)
User 3 =
(0,7,0,0,5,...,9)
User 13435 =
(0,8,0,0,0,...,7)

	User 1	User 2	User 3	User 4	...	User 13435
Airplane!	9	6		7		
Akira		4	7	8		8
Aladdin	6			7		
Alexander Nevsky				6		
...			5			
Zoolander			9	5		7

Compare corr(User 1, User 13435) to corr(User 3, User 13435).
Higher correlation → more similar!

Cosine similarity

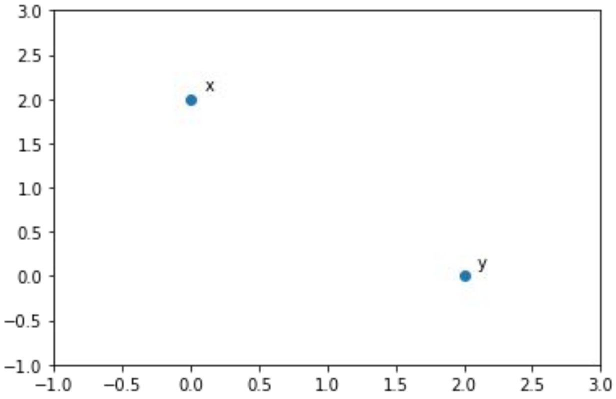


https://www.learndatasci.com/glossary/cosine-similarity/

Similarity example

Assume we only have two dimensions (movies) of data

User A = (0,2)
User B = (2,0)

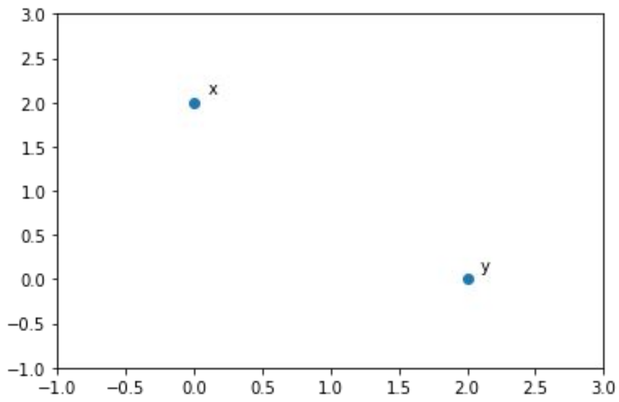


Similarity example

Assume we only have two dimensions (movies) of data

User A = (0,2)
User B = (2,0)

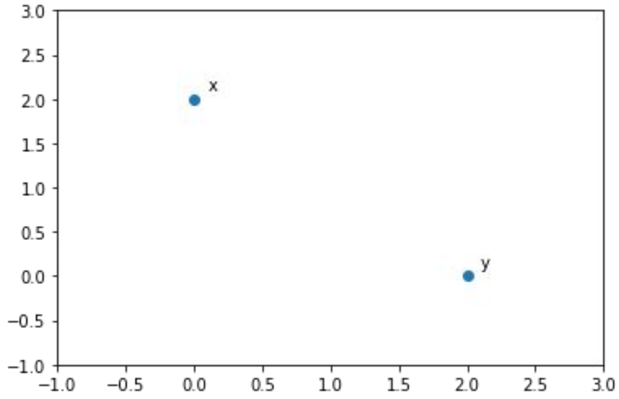
Both Pearson and Spearman correlations are -1 (practice thinking about this at home!)



Similarity example

Assume we only have two dimensions (movies) of data

User A = (0,2)
User B = (2,0)

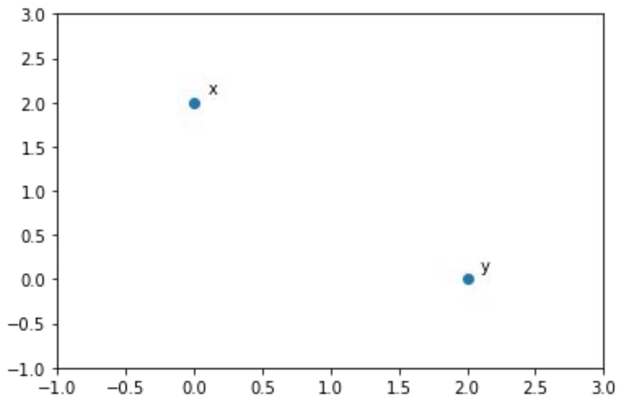


Are these two users similar based on “cosine similarity”?

Similarity example

Assume we only have two dimensions (movies) of data

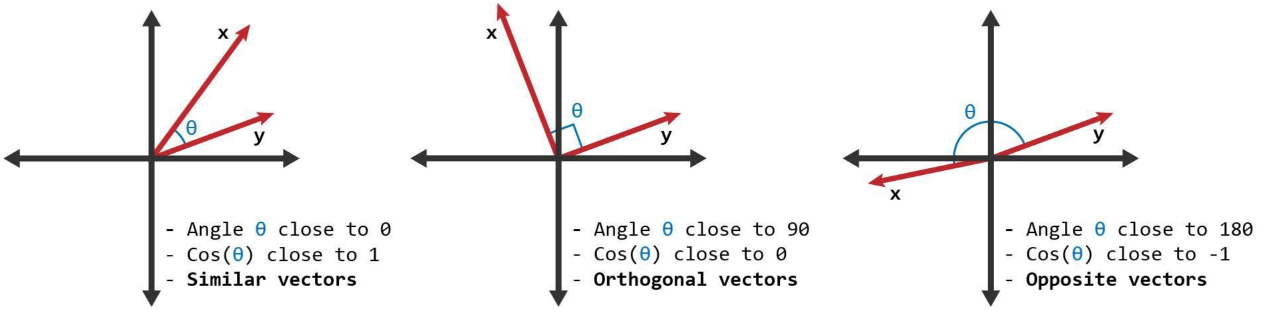
User A = (0,2)
User B = (2,0)



Calculate “cosine similarity”!

$$similarity(A,B) = cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$

Cosine similarity →

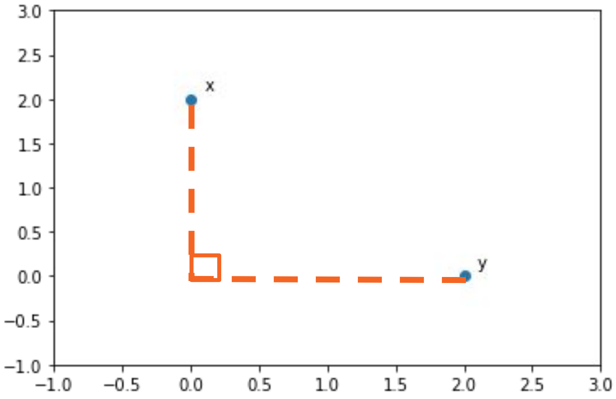


https://www.learndatasci.com/glossary/cosine-similarity/

Similarity example

Assume we only have two dimensions (movies) of data

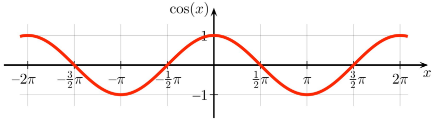
User A = (0,2)
User B = (2,0)



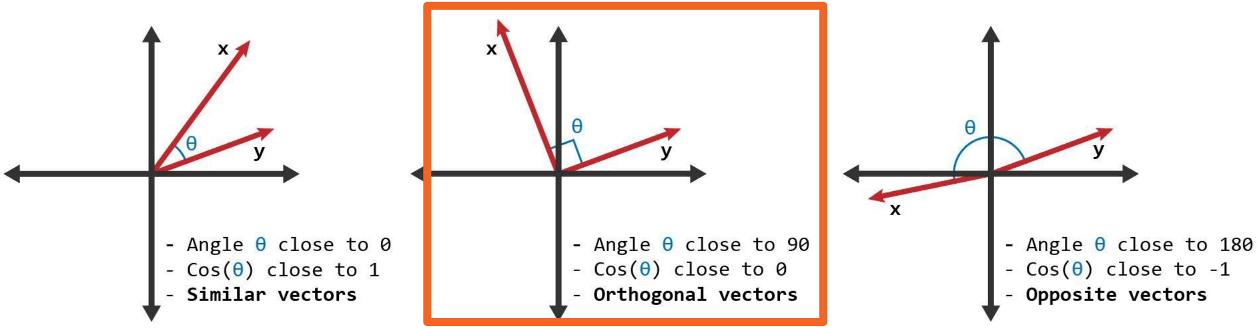
Calculate “cosine similarity”!

$$similarity(A, B) = cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$

cos(90°) = 0



Cosine similarity →

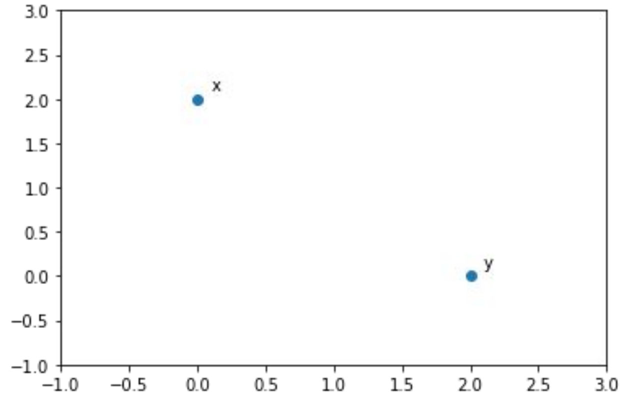


https://commons.wikimedia.org/wiki/File:Cosine.svg https://www.learndatasci.com/glossary/cosine-similarity/

Similarity example

Assume we only have two dimensions (movies) of data

User A = (0,2)
User B = (2,0)

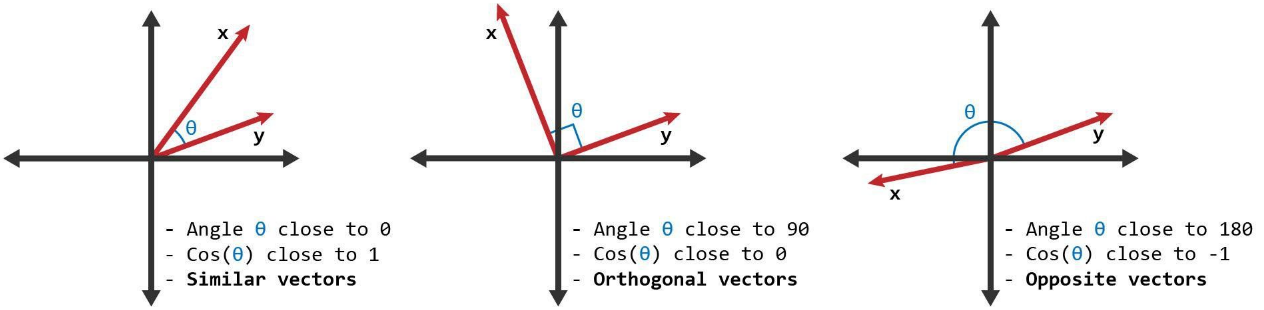


Calculate "cosine similarity"!

$$similarity(A,B) = cos(\theta) = \frac{\boxed{A \cdot B}}{\|A\|\|B\|}$$

Dot product of (0,2)·(2,0)
= (0*2 + 2*0)
(Multiply the i^{th} elements together and sum)

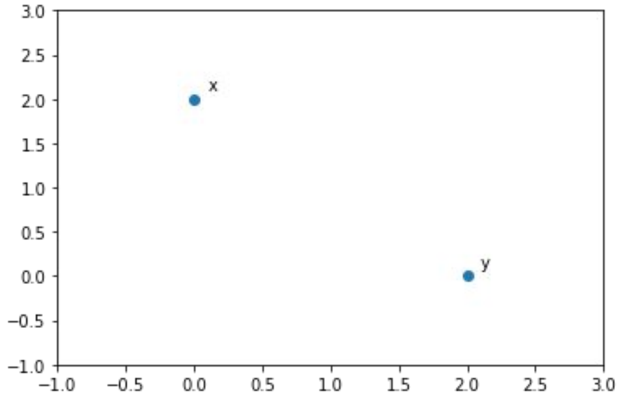
Cosine similarity →



Similarity example

Assume we only have two dimensions (movies) of data

User A = (0,2)
User B = (2,0)

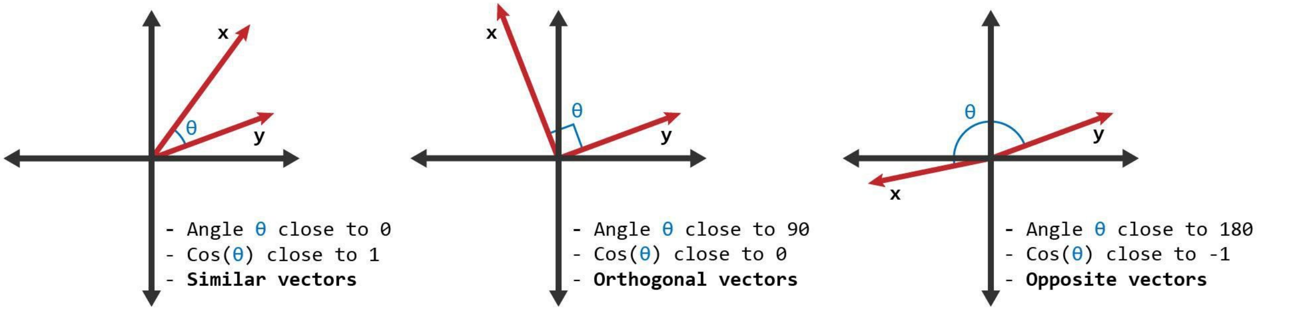


Calculate "cosine similarity"!

$$similarity(A,B) = cos(\theta) = \boxed{\frac{A \cdot B}{\|A\|\|B\|}}$$

cos(90°) = 0
= (0,2)˙(2,0) / [√(0^2+2^2)√(2^2+0^2)]
= (0*2 + 2*0) / (√4*√4) = 0 / 4 = 0

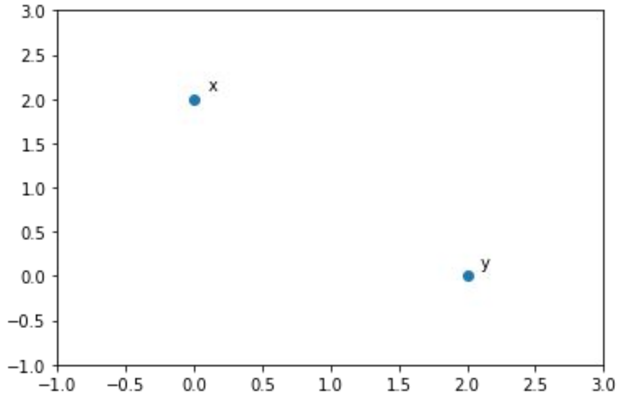
Cosine similarity →



Similarity example

Assume we only have two dimensions (movies) of data

User A = (0,2)
User B = (2,0)



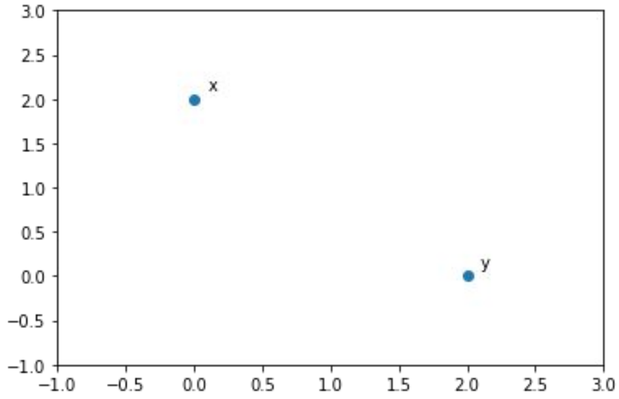
Calculate “cosine distance”!

Hint: “cosine similarity” = 1 - “cosine distance”

Similarity example

Assume we only have two dimensions (movies) of data

User A = (0,2)
User B = (2,0)



Calculate “cosine distance”!

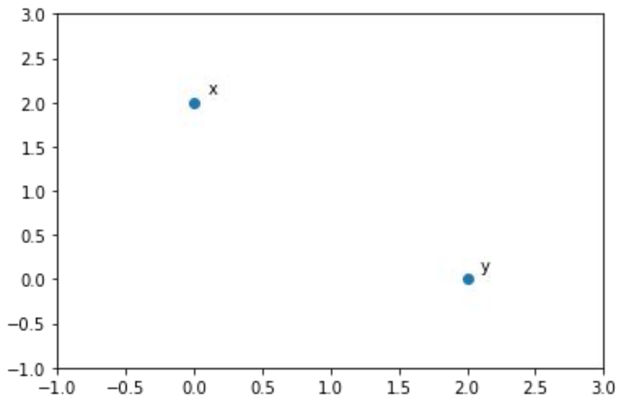
Hint: “cosine similarity” = 1 - “cosine distance”

Cosine distance
= 1 - cosine similarity
= 1 - 0 = 1

Similarity example

Assume we only have two dimensions (movies) of data

User A = (0,2)
User B = (2,0)



Cosine distance = 1

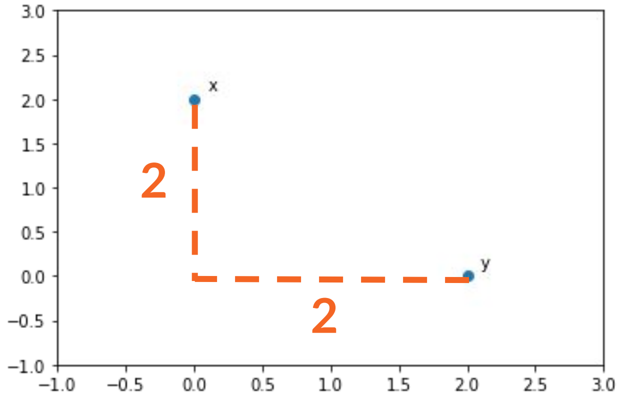
Now, calculate:

- Manhattan Distance
- Euclidean Distance

Similarity example

Assume we only have two dimensions (movies) of data

User A = (0,2)
User B = (2,0)



Cosine distance = 1

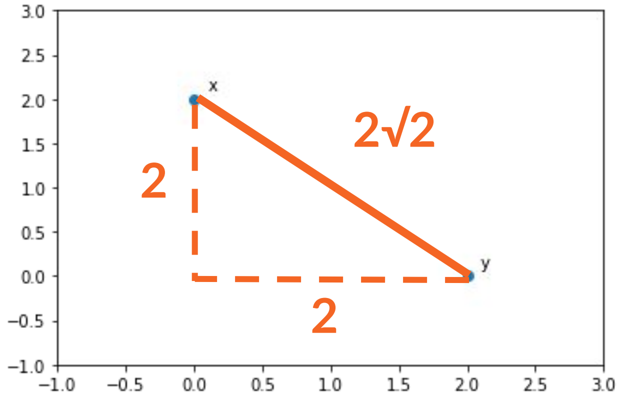
Now, calculate:

- Manhattan Distance
  2 + 2 = 4

Similarity example

Assume we only have two dimensions (movies) of data

User A = (0,2)
User B = (2,0)



Cosine distance = 1

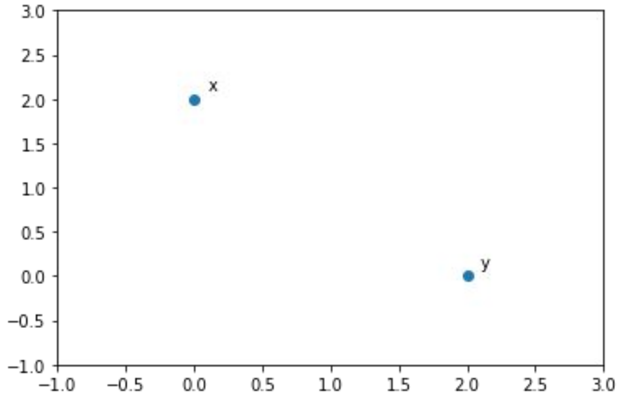
Now, calculate:

- Euclidean Distance
  $\sqrt{(2^2+2^2)} = \sqrt{8}$

Similarity example

Assume we only have two dimensions (movies) of data

User A = (0,2)
User B = (2,0)



Different distance metrics give us different distances!

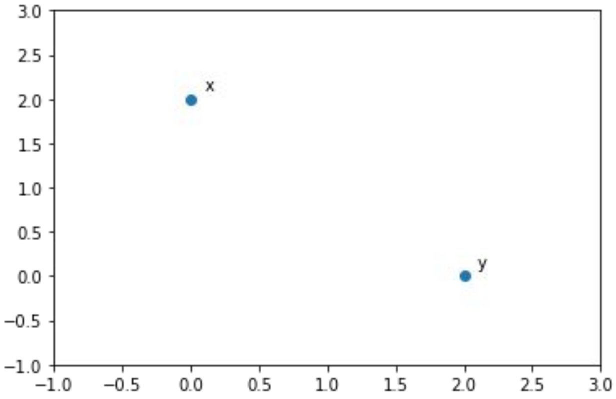
When should we choose each?

https://www.learndatasci.com/glossary/cosine-similarity/

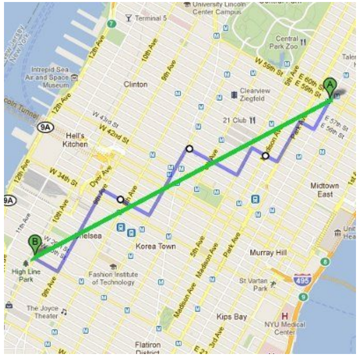
Similarity example

Assume we only have two dimensions (movies) of data

User A = (0,2)
User B = (2,0)



[Euclidean](#) vs. [Manhattan](#) distance: depends whether you’re allowed to take the “shortest possible path” in 2d space, or if you need to go along “city blocks”

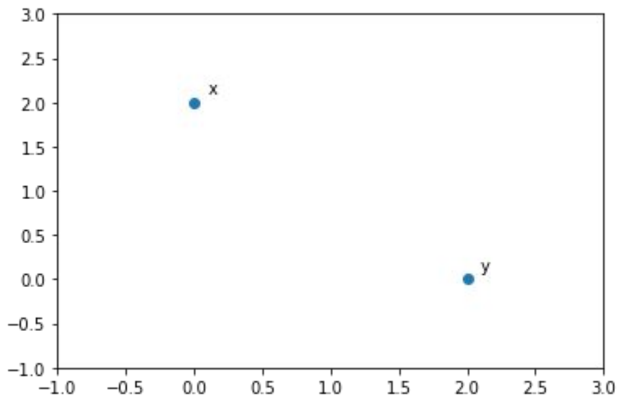


https://www.quora.com/Why-do-people-use-Euclidean-distance-instead-of-Manhattan-distance-or-some-other-norm-metric

Similarity example

Assume we only have two dimensions (movies) of data

User A = (0,2)
User B = (2,0)

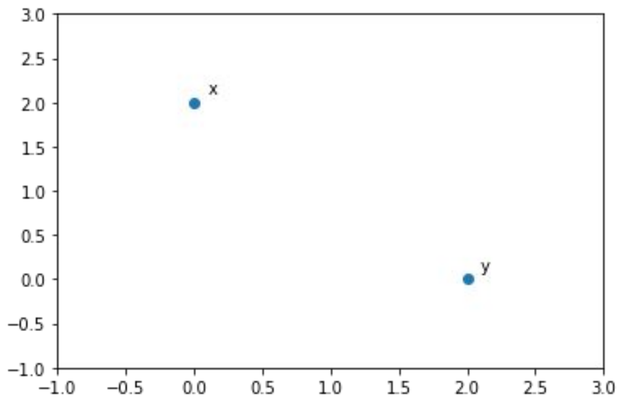


<u>Cosine distance</u>: when *magnitude* of features doesn't matter *(User A could just as well be (0,5) or (0,10) and the cosine similarity would stay the same)*, but the *angle* between vectors matters

Similarity example

Assume we only have two dimensions (movies) of data

User A = (0,2)
User B = (2,0)



Cosine distance: when *magnitude* of features doesn't matter *(User A could just as well be (0,5) or (0,10) and the cosine similarity would stay the same)*, but the *angle* between vectors matters

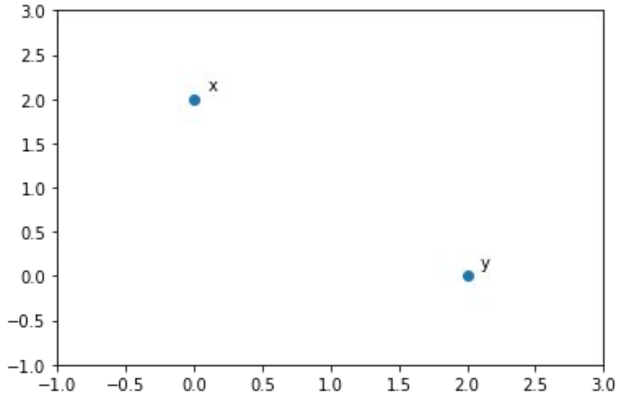
Generally, cosine distance is preferred in high dimensions because it only cares about the terms both vectors have in common (else dot product = 0)

https://www.learndatasci.com/glossary/cosine-similarity/

Similarity example

Assume we only have two dimensions (movies) of data

User A = (0,2)
User B = (2,0)



Cosine distance: when *magnitude* of features doesn't matter *(User A could just as well be (0,5) or (0,10) and the cosine similarity would stay the same)*, but the *angle* between vectors matters

User 3 =
(0,7,0,0,5,...,9)
User 13435 =
(0,8,0,0,0,...,7)

Generally, cosine distance is preferred in high dimensions because it only cares about the terms both vectors have in common (else dot product = 0)

https://www.learndatasci.com/glossary/cosine-similarity/

One way: find other people who liked similar movies

Step 1. Find a set of users whose ratings are “similar” to User 13435

Similarity can be calculated with metrics like Pearson correlation, or “cosine similarity” = 1 - “cosine distance”

	User 1	User 2	User 3	User 4	...	User 13435
Airplane!	9	6		7		
Akira		4	7	8		8
Aladdin	6			7		
Alexander Nevsky				6		
...			5			
Zoolander			9	5		7

One way: find other people who liked similar movies

Step 2. Estimate User 13435's ratings for movies they *haven't* rated by using similar users' ratings as a proxy

	User 1	User 2	User 3	User 4	...	User 13435
Airplane!	9	6		7		
Akira		4	7	8		8
Aladdin	6			7		
Alexander Nevsky				6		
...			$\boxed{5}$			5?
Zoolander			9	5		7

One way: find other people who liked similar movies

Step 2. Estimate User 13435's ratings for movies they *haven't* rated by using similar users' ratings as a proxy

Calculate a weighted average of ratings (weighted by similarity of users to User 13435)

	User 1	User 2	User 3	User 4	...	User 13435
Airplane!	9	6		7		
Akira		4	7	8		8
Aladdin	6			7		
Alexander Nevsky				6		
...			5			5?
Zoolander			9	5		7

One way: find other people who liked similar movies

This process is called "user-user collaborative filtering"

	User 1	User 2	User 3	User 4	...	User 13435
Airplane!	9	6		7		
Akira		4	7	8		8
Aladdin	6			7		
Alexander Nevsky				6		
...						
Zoolander			9	5		7

Movie rating data

How else can we use this dataset to recommend a movie to a specific user?

	User 1	User 2	User 3	User 4	...	User 13435
Airplane!	9	6		7		
Akira		4	7	8		8
Aladdin	6			7		
Alexander Nevsky				6		?
...						
Zoolander			9	5		7

Another way: find other movies similar to ones you liked

We can also do "item-item collaborative filtering"

	User 1	User 2	User 3	User 4	...	User 13435
Airplane!	9	6		7		
Akira		4	7	8		8
Aladdin	6			7		
Alexander Nevsky				6		
...						
Zoolander			9	5		7

Another way: find other movies similar to ones you liked

Step 1. For a movie User 13435 hasn't watched, calculate similarities with movies that User 13435 *has* watched

Do this by using *other user's* ratings

	User 1	User 2	User 3	User 4	...	User 13435
Airplane!	9	6		7		
Akira		4	7	8		8
Aladdin	6			7		
Alexander Nevsky				6		?
...						
Zoolander			9	5		7

Another way: find other movies similar to ones you liked

	User 1	User 2	User 3	User 4	...	User 13435
Airplane!	9	6		7		
Akira		4	7	8		8
Aladdin	6			7		
Alexander Nevsky				6		?
...						
Zoolander			9	5		7

Some low similarity score

A slightly higher similarity score

Another way: find other movies similar to ones you liked

Step 2. Estimate the rating for User 13435 by calculating a weighted average of User 13435’s ratings for other movies, weighted by similarity

	User 1	User 2	User 3	User 4	...	User 13435
Airplane!	9	6		7		
Akira		4	7	8		8
Aladdin	6			7		
Alexander Nevsky				6		?
...						
Zoolander			9	5		7

Collaborative Filtering

User-User

$$r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}}$$

Item-Item

$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

Collaborative Filtering

User-User

$$r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}}$$

Item-Item

$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

We want to predict the rating *r* for item *i* (a movie) for user *x*

- Item i
- User x

Collaborative Filtering

User-User

$$r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}}$$

Looks across different users *y*

Item-Item

$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

Looks across our items *j* (movies user x has watched and rated)

- Item i
- User x
- Other items j
- Other users y

Collaborative Filtering

User-User

$$r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}}$$

Similarity between our user x and other user y

Item-Item

$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

Similarity between movie i and our movie j

- Item i
- User x
- Other items j
- Other users y

Collaborative Filtering

User-User

$$r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}}$$

Rating for item *i* by other user *y*

Item-Item

$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

Rating by user *x* for our movie *j*

- Item i
- User x
- Other items j
- Other users y

Collaborative Filtering

User-User

$$r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}}$$

Weighted average of ratings for item *i* across similar users *y*

Item-Item

$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

Weighted average of ratings for user *x* across our items *j*

Collaborative Filtering

User-User

$$r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}}$$

Item-Item

$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

Which is better in practice?

Collaborative Filtering

User-User

$$r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}}$$

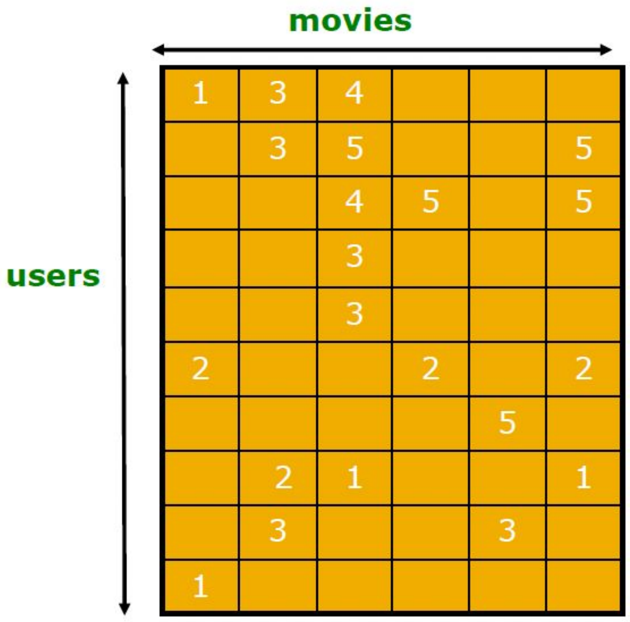
Item-Item

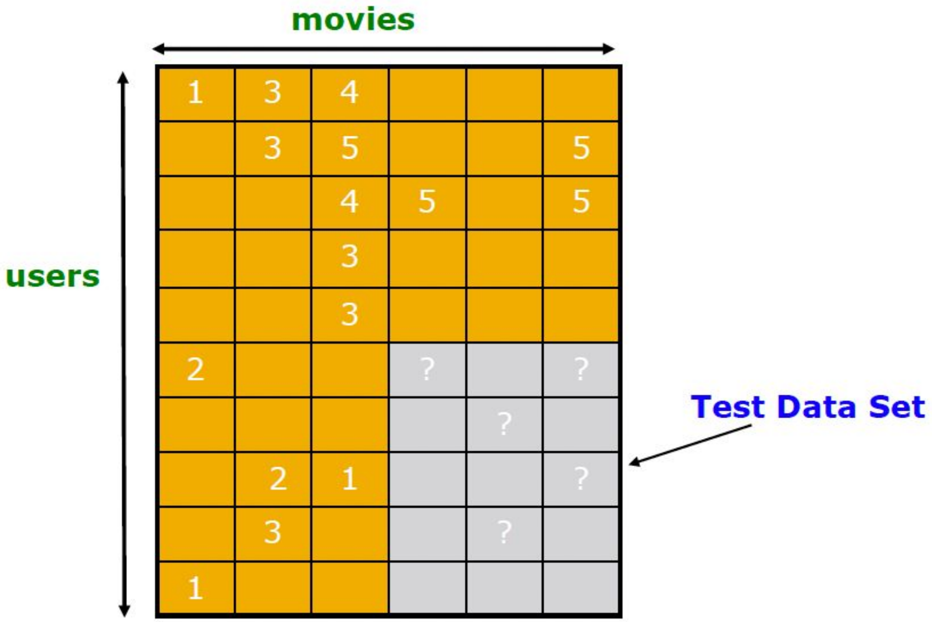
$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

Which is better in practice?

People tend to use item-item more because users have different tastes, but it always depends!

Collaborative Filtering as a model





Collaborative Filtering as a model

- Evaluation metrics depend on application
- Some common ones for rec sys:
  - **RMSE** (Root mean square error) if you know true ratings for r_{xi}
  - Precision in top *n* rated movies
  - ROC if binary outcome

Collaborative Filtering

Why might we *not* want to use CF?

- **Sparsity**: you need enough data in the system to find similar users/movies in the first place!
  - **Cold Start:** Can't recommend a new, unrated item
  - Can be hard to find users who have rated the same items

Collaborative Filtering

Why might we *not* want to use CF?

- **Sparsity**: you need enough data in the system to find similar users/movies in the first place!
  - **Cold Start:** Can't recommend a new, unrated item
  - Can be hard to find users who have rated the same items
- **Popularity bias:** CF tends to recommend popular items, which is not necessarily good for niche taste

Collaborative Filtering

Why might we *not* want to use CF?

- **Sparsity**: you need enough data in the system to find similar users/movies in the first place!
  - **Cold Start:** Can't recommend a new, unrated item
  - Can be hard to find users who have rated the same items
- **Popularity bias:** CF tends to recommend popular items, which is not necessarily good for niche taste
- **Costly**: takes a long time to find most similar users

Collaborative Filtering

What if there's a *method* that tackles the user-item recommendations using a *smaller representation*?

1 minute break

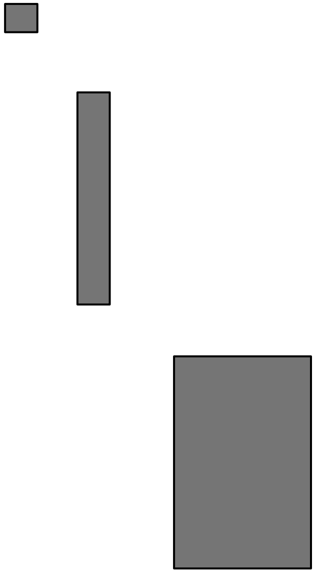
https://rpubs.com/Argaadya/recommender-svdf

Linear algebra

A single number is a *scalar*

A one-dimensional array is a *vector*

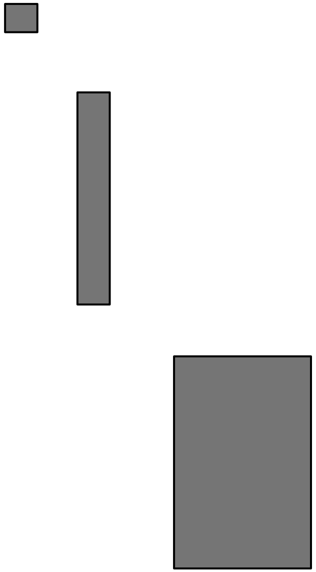
A two-dimensional array is a *matrix*



Linear algebra

To build up to understanding Singular Value Decomposition, we just have to understand how to *multiply* these things.

It's really just arithmetic!



Vectors

Column vector

2
0
0
1
0

Row vector

2	0	0	1	0

Vectors

Column vector

$$x = \begin{bmatrix} 2 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Row vector

$$x^T = \begin{bmatrix} 2 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Vectors

Column vector

$x =$

2
0
0
1
0

Row vector

$x^T =$

2	0	0	1	0

"x transpose"

Vectors

Column vector

$$x = \begin{bmatrix} 2 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

5x1

Row vector

$$x^T = \begin{bmatrix} 2 & 0 & 0 & 1 & 0 \end{bmatrix}$$

1x5

Multiplying vectors

y

5x1 matrix

$$y = \begin{bmatrix} 2 \\ 0 \\ 0 \\ 1 \\ 3 \end{bmatrix}$$

x

$$x = \begin{bmatrix} 1 & 2 & 1 & 0 & 1 \end{bmatrix} \qquad \boxed{?}$$

1x5 matrix

Version 1: inner product (dot product) $(x \cdot y)$

Multiplying vectors

2
0
0
1
3

1	2	1	0	1

2

Multiplying vectors

2
0
0
1
3

1	**2**	1	0	1

2+0

Multiplying vectors

2
0
0
1
3

1	2	**1**	0	1

2+0+0

Multiplying vectors

$$\begin{bmatrix} 2 \\ 0 \\ 0 \\ \mathbf{1} \\ 3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 1 & \mathbf{0} & 1 \end{bmatrix} \qquad \boxed{\mathbf{2+0+0+0}}$$

Multiplying vectors

$$\begin{bmatrix} 2 \\ 0 \\ 0 \\ 1 \\ \mathbf{3} \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 1 & 0 & \mathbf{1} \end{bmatrix} \qquad \boxed{\mathbf{2+0+0+0+3}}$$

Multiplying vectors

y

5x1 matrix

$$\begin{bmatrix} 2 \\ 0 \\ 0 \\ 1 \\ 3 \end{bmatrix}$$

x

1x5 matrix

$$\begin{bmatrix} 1 & 2 & 1 & 0 & 1 \end{bmatrix} \quad \boxed{\mathbf{5}}$$

← dot product of x and y has size 1x1 (i.e. is a scalar)

Multiplying vectors

2
0
-2
1
-1

2	2	-1	0	3

?

Multiplying vectors

2
0
-2
1
-1

2	2	-1	0	3

4 + 0 + 2 + 0 + -3 = **3**

Multiplying vectors

2
0
0
1
0

2	0	0	1	0

?

What is the dot product $x \cdot x$?

Multiplying vectors

2
0
0
1
0

2	0	0	1	0

$2^2+1^2=$**5**

Inner product of a vector with itself is the **sum of squared entries**

Multiplying vectors

2
0
0
1
0

0	2	1	0	0

?

What is the dot product?

Multiplying vectors

$$\begin{bmatrix} 0 & 2 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \boxed{0}$$

If the inner product is 0, the two vectors are **orthogonal**

Multiplying vectors

Inner product is $\mathbf{x}^T\mathbf{x}$
("x transpose x")

$$\mathbf{x} = \begin{bmatrix} 2 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

$$\mathbf{x}^T = \begin{bmatrix} 0 & 2 & 1 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 \end{bmatrix}$$

Multiplying vectors

x^T

x

	0	2	1	0	0
2					
0					
0					
1					
0					

Version 2: outer product

Multiplying vectors

x^T

0	2	1	0	0

x

2
0
0
1
0

Version 2: outer product

Outer product is xx^T ("x x transpose")

Multiplying vectors

	0	2	1	0	0
2	0				
0					
0					
1					
0					

Multiplying vectors

0	2	1	0	0

2
0
0
1
0

0	4	2	0	0

Multiplying vectors

0	2	1	0	0

2
0
0
1
0

0	4	2	0	0
0	0	0	0	0
0	0	0	0	0
0	2	1	0	0
0	0	0	0	0

Multiplying vectors

0	2	1	0	0

2
0
0
1
0

0	4	2	0	0
0	**0**	**0**	**0**	**0**
0	0	0	0	0
0	2	1	0	0
0	0	0	0	0

A zero will result in a whole row of zeroes

Multiplying vectors

0	**2**	**1**	0	0

2
0
0
1
0

0	4	2	0	0
0	0	0	0	0
0	0	0	0	0
0	2	1	0	0
0	0	0	0	0

Non-zeros create "stripes"

Multiplying vectors

0	**2**	**1**	0	0

2
0
0
1
0

0	4	2	0	0
0	0	0	0	0
0	0	0	0	0
0	2	1	0	0
0	0	0	0	0

Non-zeros are at intersections

Multiplying vectors

1	1	0	2	1

0
2
0
1
0

Multiplying vectors

1	1	0	2	1

0
2
0
1
0

0	0	0	0	0
		0		
0	0	0	0	0
		0		
0	0	0	0	0

Multiplying vectors

1	1	0	2	1

0
2
0
1
0

0	0	0	0	0
?	?	0	?	?
0	0	0	0	0
		0		
0	0	0	0	0

Fill in the blanks of the outer product

Multiplying vectors

1	1	0	2	1

0
2
0
1
0

0	0	0	0	0
2	2	0	4	2
0	0	0	0	0
		0		
0	0	0	0	0

Multiplying vectors

1	1	0	2	1

0
2
0
1
0

0	0	0	0	0
2	2	0	4	2
0	0	0	0	0
1	1	0	2	1
0	0	0	0	0

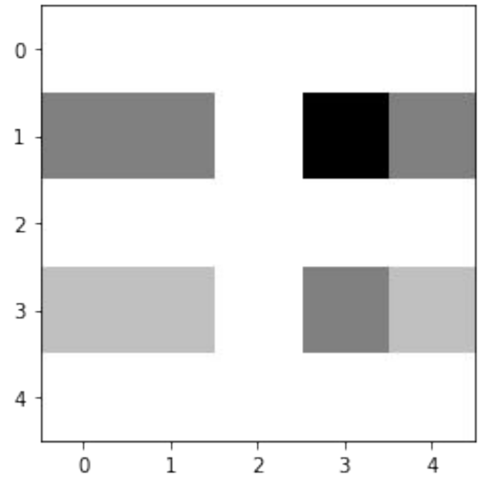
Multiplying vectors

	1	1	0	2	1
0					
2	2	2		4	2
0					
1	1	1		2	1
0					

Most of the resulting matrix is 0's

Visualize a matrix as an image

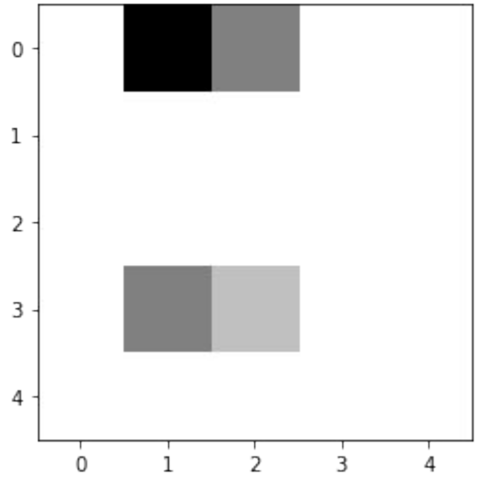
2	2		4	2
1	1		2	1



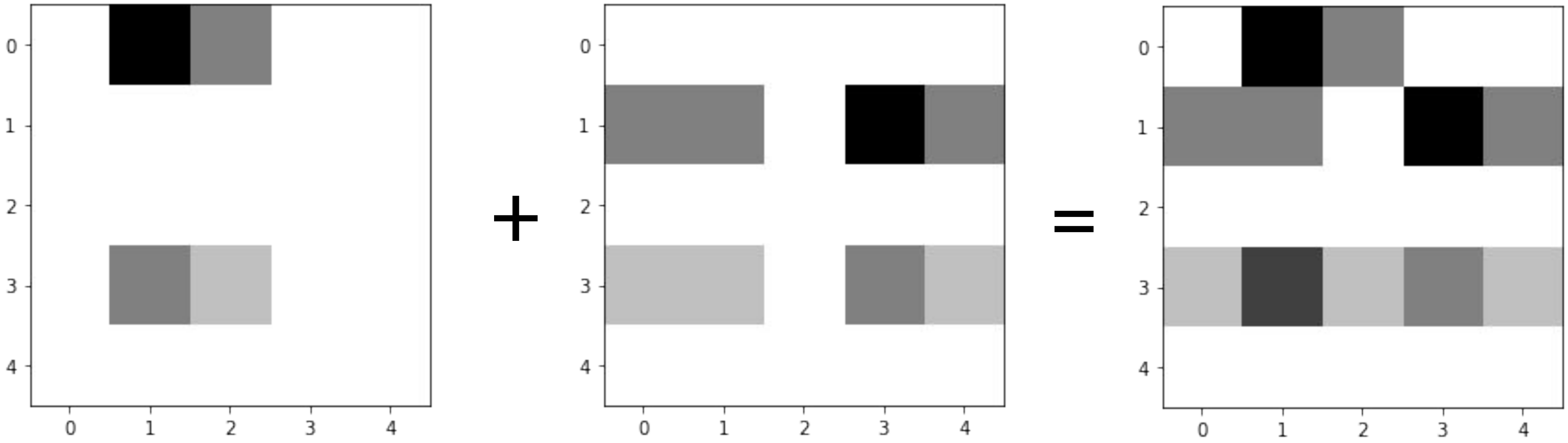
The cells with non-zeros are shaded proportional to their values

Visualize a matrix as an image

0	4	2	0	0
0	0	0	0	0
0	0	0	0	0
0	2	1	0	0
0	0	0	0	0



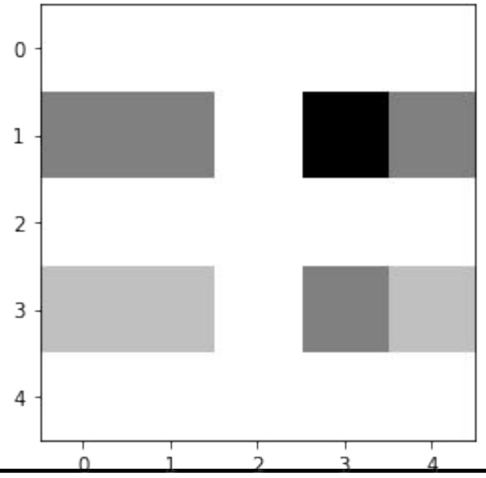
Adding matrices



Side note: matrix storage in computers

How do we actually store matrix A in Python?

2	2		4	2
1	1		2	1



Side note: matrix storage in computers

We’re used to doing something like…

2	2		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

Side note: matrix storage in computers

But computers have limited memory, and we're wasting a lot of resources by typing a lot of 0's in "sparse matrices"!

2	2		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

Side note: matrix storage in computers

Instead of typing out all the numbers in this format (which requires remembering 25 numbers), we can instead use multiple arrays for *smaller storage!*

2	2		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

Sparse matrix storage

What if we only stored the number of non-zeroes, and each of their locations (col, row)?

2	2		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

https://en.wikipedia.org/wiki/Sparse_matrix

Sparse matrix storage

What is the number of non-zeroes (nnz) in this matrix?

2	2		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

https://en.wikipedia.org/wiki/Sparse_matrix

Sparse matrix storage

There are 8 non-zero numbers (2,2,4,2,1,1,2,1) and their location in the matrix can each be expressed with their row and column indices!

2	2		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

https://en.wikipedia.org/wiki/Sparse_matrix

Coordinate List (COO) format

We store our matrix values in 3 separate arrays!

Value =
Column_Index =
Row_Index =

This is the format you'll use for #B7 of HW6.

2	2		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

https://en.wikipedia.org/wiki/Sparse_matrix

Coordinate List (COO) format

Value = [2
Column_Index = [0
Row_Index = [1

The first non-zero value appears in column 0 row 1. Its value is 2.

2	2		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

Coordinate List (COO) format

Value = [2, 2
Column_Index = [0, 1
Row_Index = [1, 1

The second non-zero value appears in column 1 row 1. Its value is 2.

2	**2**		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

Coordinate List (COO) format

Value = [2, 2, 4, 2, 1, 1, 2, __]
Column_Index = [0, 1, 3, 4, 0, 1, 3, __]
Row_Index = [1, 1, 1, 1, 3, 3, 3, __]

We can do this for all the non-zero numbers. Fill in the last one!

2	2		4	2
1	1		2	**1**

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

Coordinate List (COO) format

Value = [2, 2, 4, 2, 1, 1, 2, 1]
Column_Index = [0, 1, 3, 4, 0, 1, 3, 4]
Row_Index = [1, 1, 1, 1, 3, 3, 3, 3]

2	2		4	2
1	1		2	**1**

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

The highlighted value of 1 is in row 3, column 4

Coordinate List (COO) format

Length 8 → Value = [2, 2, 4, 2, 1, 1, 2, 1]

Length 8 → Column_Index = [0, 1, 3, 4, 0, 1, 3, 4]

Length 8 → Row_Index = [1, 1, 1, 1, 3, 3, 3, 3]

The length of each of these three arrays is simply the nnz = 8

2	2		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

Coordinate List (COO) format

Length 8 → Value = [2, 2, 4, 2, 1, 1, 2, 1]

Length 8 → Column_Index = [0, 1, 3, 4, 0, 1, 3, 4]

Length 8 → Row_Index = [1, 1, 1, 1, 3, 3, 3, 3]

We can now store the same matrix information in 3*8 = 24 numbers!

2	2		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

Coordinate List (COO) format

Length 8 → Value = [2, 2, 4, 2, 1, 1, 2, 1]
Length 8 → Column_Index = [0, 1, 3, 4, 0, 1, 3, 4]
Length 8 → Row_Index = [1, 1, 1, 1, 3, 3, 3, 3]

We can now store the same matrix information in 3*8 = 24 numbers!

2	2		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
               [2,2,0,4,2],
               [0,0,0,0,0],
               [1,1,0,2,1],
               [0,0,0,0,0]])
```

In contrast, this was 25 #s!

Compressed Sparse Row format

Another format: we can actually do *even better* on compressing storage!

Value =
Column_Index =
Row_Index =

The only difference is in how the *row index* is stored.

2	2		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

https://en.wikipedia.org/wiki/Sparse_matrix

Compressed Sparse Row format

Value = [2

Column_Index = [0

Row_Index = [0

The first non-zero value appears in column 0 row 1. Its value is 2.

2	2		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

Compressed Sparse Row format

The Row_Index always starts with 0. Then, it tells you a running tally of how many non-zero elements exist, updating for each row.

Value = [2
Column_Index = [0
Row_Index = [0

2	2		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

Compressed Sparse Row format

The Row_Index always starts with 0. Then, it tells you a running tally of how many non-zero elements exist, updating for each row.

In the 0th row, there are <u>0</u> non-zero elements

Value = [2

Column_Index = [0

Row_Index = [0 , <u>0</u>

2	2		4	2
1	1		2	1

A = np.array([[0,0,0,0,0],
[2,2,0,4,2],
[0,0,0,0,0],
[1,1,0,2,1],
[0,0,0,0,0]])

Compressed Sparse Row format

The next non-zero value appears in column 1 row 1; its value is also 2. We can update Value and Column_Index.

We haven’t reached the end of our row yet, so we don’t update Row_Index

Value = [2, 2
Column_Index = [0, 1
Row_Index = [0, 0

2	**2**		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

Compressed Sparse Row format

Value = [2, 2, 4, 2,
Column_Index = [0, 1, 3, 4,
Row_Index = [0, 0, 4,

At the end of the 2nd row of [2,2,0,4,2], we can update Row_Index to include 4 since there are 4 non-zero values in the 2nd row

2	2		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

Compressed Sparse Row format

Value = [2, 2, 4, 2,
Column_Index = [0, 1, 3, 4,
Row_Index = [0, 0, 4, 4,

At the end of the 3rd row (highlighted), we can update Row_Index to include another 4 (since our running count of non-zero numbers doesn't change)

2	2		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

Compressed Sparse Row format

Value = [2, 2, 4, 2, ___

Column_Index = [0, 1, 3, 4, ___

Row_Index = [0, 0, 4, 4,

Fill in the blanks for this cell

2	2		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

Compressed Sparse Row format

Value = [2, 2, 4, 2, 1

Column_Index = [0, 1, 3, 4, 0

Row_Index = [0, 0, 4, 4,

2	2		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

Value is 1 in the 0th column. We aren't at the end of the row yet

Compressed Sparse Row format

Value = [2, 2, 4, 2, 1, 1, 2, 1]
Column_Index = [0, 1, 3, 4, 0, 1, 3, 4]
Row_Index = [0, 0, 4, 4, 8, 8]

The running count of non-zero numbers is 8 at the end of row 4, and still 8 at the end of row 5 (since row 5 is all zeroes)

2	2		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

Compressed Sparse Row format

Value = [2, 2, 4, 2, 1, 1, 2, 1]
Column_Index = [0, 1, 3, 4, 0, 1, 3, 4]
Row_Index = [0, 0, 4, 4, 8, 8]

Is CSR better for compressing our matrix storage?

2	2		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

Compressed Sparse Row format

Length 8 → Value = [2, 2, 4, 2, 1, 1, 2, 1]

Length 8 → Column_Index = [0, 1, 3, 4, 0, 1, 3, 4]

Length 6 → Row_Index = [0, 0, 4, 4, 8, 8]

2	2		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

Compressed Sparse Row format

Length 8 → Value = [2, 2, 4, 2, 1, 1, 2, 1]
Length 8 → Column_Index = [0, 1, 3, 4, 0, 1, 3, 4]
Length 6 → Row_Index = [0, 0, 4, 4, 8, 8]

rows + 1 (depending on matrix size and sparsity, this could be smaller than nnz)!

2	2		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

Compressed Sparse Row format

Length 8 → Value = [2, 2, 4, 2, 1, 1, 2, 1]

Length 8 → Column_Index = [0, 1, 3, 4, 0, 1, 3, 4]

Length 6 → Row_Index = [0, 0, 4, 4, 8, 8]

Now we only have to store 8+8+6=22 numbers instead of 25!

2	2		4	2
1	1		2	1

```
A = np.array([[0,0,0,0,0],
              [2,2,0,4,2],
              [0,0,0,0,0],
              [1,1,0,2,1],
              [0,0,0,0,0]])
```

Quick note on HW

- HW6 C12 asks you to find the KMeans cluster number, where our given output answer is 97
- Some students have been getting a cluster number of 84 instead. This appears to be a version difference!
- We're using sklearn 1.3.2. If you update your sklearn version to match ours, you should also get a cluster number of 97!

Compressed Sparse Row format

It's efficient to store sparse matrices (with many zeroes) in CSR format, and this is generally done for massive matrices in practice!

scipy.sparse.csr_matrix

```
class scipy.sparse.csr_matrix(arg1, shape=None, dtype=None, copy=False)
```

Compressed Sparse Row matrix

https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html

Compressed Sparse Row format

```
from scipy.sparse import csr_matrix

A = np.array([[0, 0, 0, 0, 0],
              [2, 2, 0, 4, 2],
              [0, 0, 0, 0, 0],
              [1, 1, 0, 2, 1],
              [0, 0, 0, 0, 0]])

# Convert the matrix to CSR format
A_csr = csr_matrix(A)

# Print the CSR format arrays
print("Data array (A.data):", A_csr.data)
print("Indices array (A.indices):", A_csr.indices)
print("Indptr array (A.indptr):", A_csr.indptr)
```

```
Data array (A.data): [2 2 4 2 1 1 2 1]
Indices array (A.indices): [0 1 3 4 0 1 3 4]
Indptr array (A.indptr): [0 0 4 4 8 8]
```

https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html

1 minute break & attendance

tinyurl.com/38vyy9wb

Multiplying matrices

B
(2x5 matrix)

0	2	1	0	0
1	1	0	2	1

A
(5x2 matrix)

2	0
0	2
0	0
1	1
0	0

Multiplying matrices

0	2	1	0	0
1	1	0	2	1

B
(2x5 matrix)

A
(5x2 matrix)

2	0
0	2
0	0
1	1
0	0

Multiply to make
AB
(5x5 matrix)

Multiplying matrices

0	2	1	0	0
1	1	0	2	1

B
(2x5 matrix)

A
(5x2 matrix)

2	**0**
0	2
0	0
1	1
0	0

0				

Each cell is the inner product (dot product) of two vectors!

2*0+0*1=0

Multiplying matrices

0	**2**	1	0	0
1	**1**	0	2	1

B
(2x5 matrix)

A
(5x2 matrix)

2	**0**
0	2
0	0
1	1
0	0

0	4			

Each cell is the inner product (dot product) of two vectors!

2*2+0*1=4

Multiplying matrices

B
(2x5 matrix)

0	2	1	0	0
1	1	0	2	1

A
(5x2 matrix)

2	0
0	2
0	0
1	1
0	0

	4	2		
2	?		4	2
1	3	1	2	1

What is this cell of AB?

Multiplying matrices

B
(2x5 matrix)

0	2	1	0	0
1	1	0	2	1

A
(5x2 matrix)

2	0
0	2
0	0
1	1
0	0

	4	2		
2	2		4	2
1	3	1	2	1

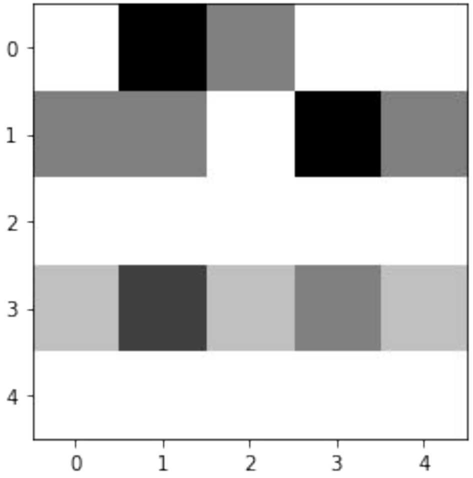
0*2+2*1 = 2

We can visualize AB the same way

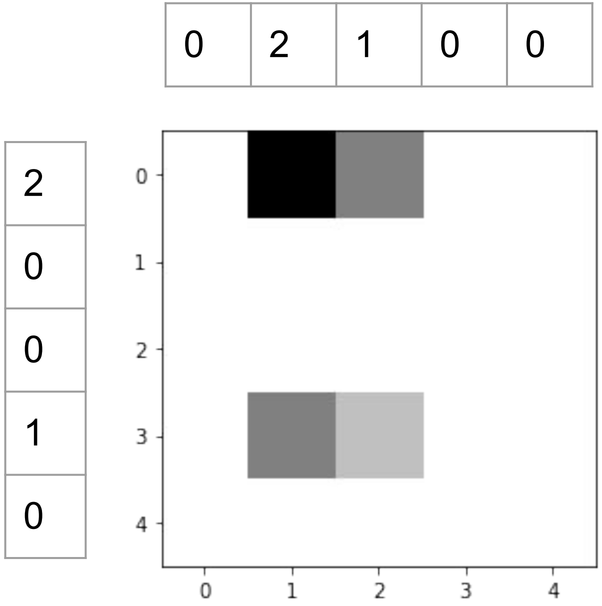
0	2	1	0	0
1	1	0	2	1

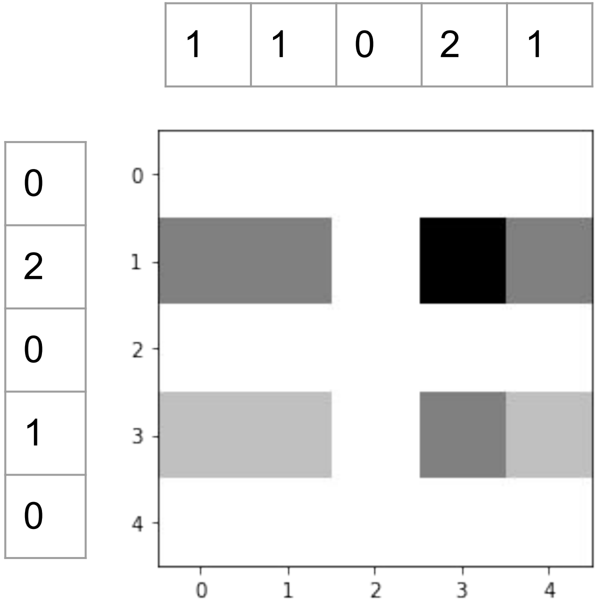
2	0
0	2
0	0
1	1
0	0

	4	2		
2	2		4	2
1	3	1	2	1

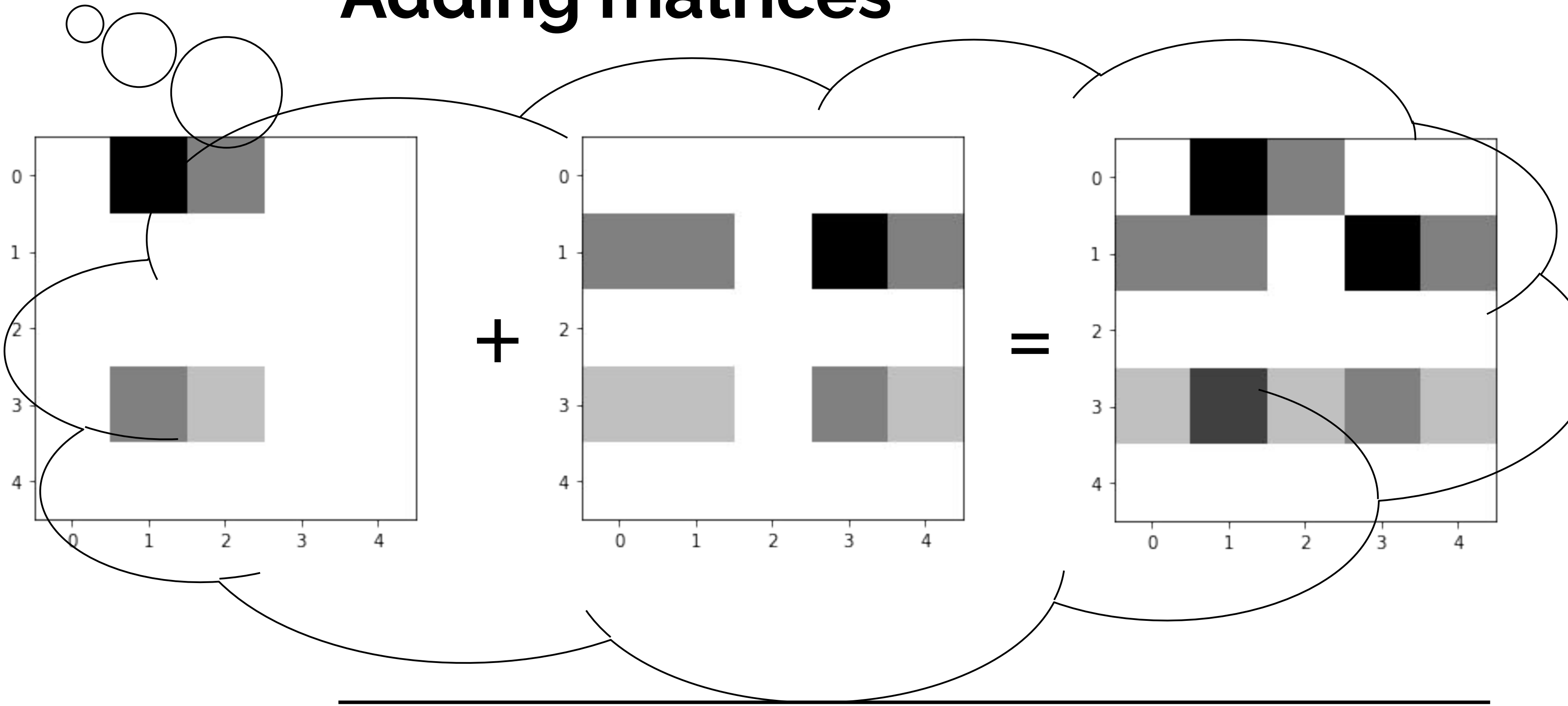


The two outer products from before…





Adding matrices



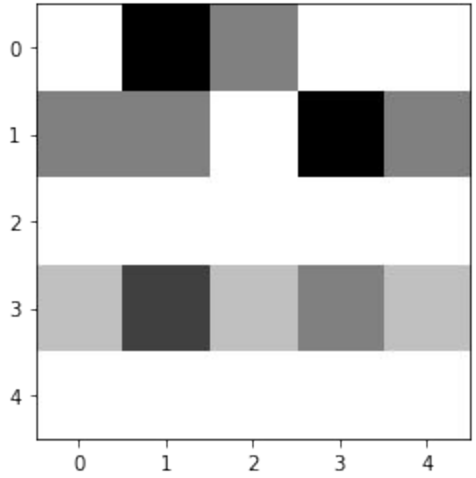
We added the matrices by multiplying concatenated vectors!

0	2	1	0	0
1	1	0	2	1

(the two row vectors, top and bottom)

2	0
0	2
0	0
1	1
0	0

	4	2		
2	2		4	2
1	3	1	2	1



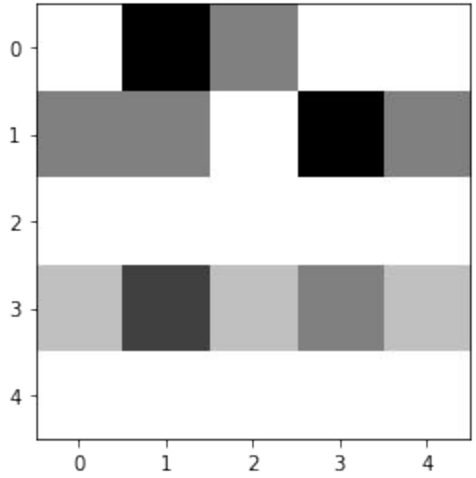
(the two column vectors, side by side)

That was a sum, but could we do a weighted sum?

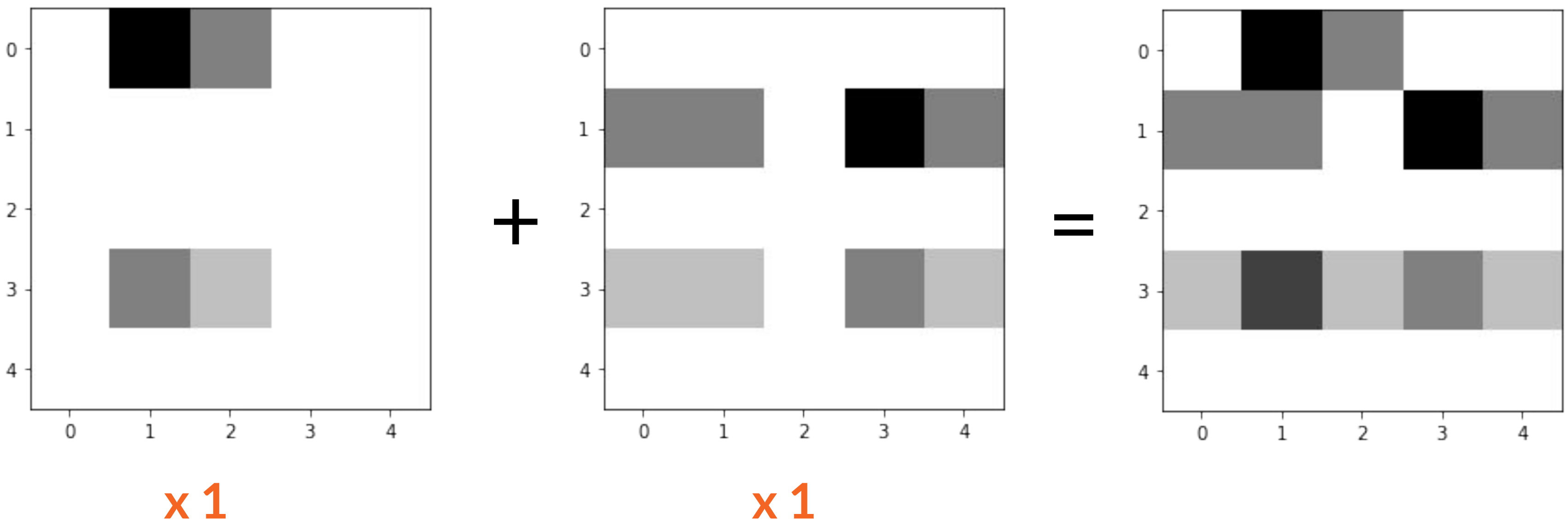
0	2	1	0	0
1	1	0	2	1

2	0
0	2
0	0
1	1
0	0

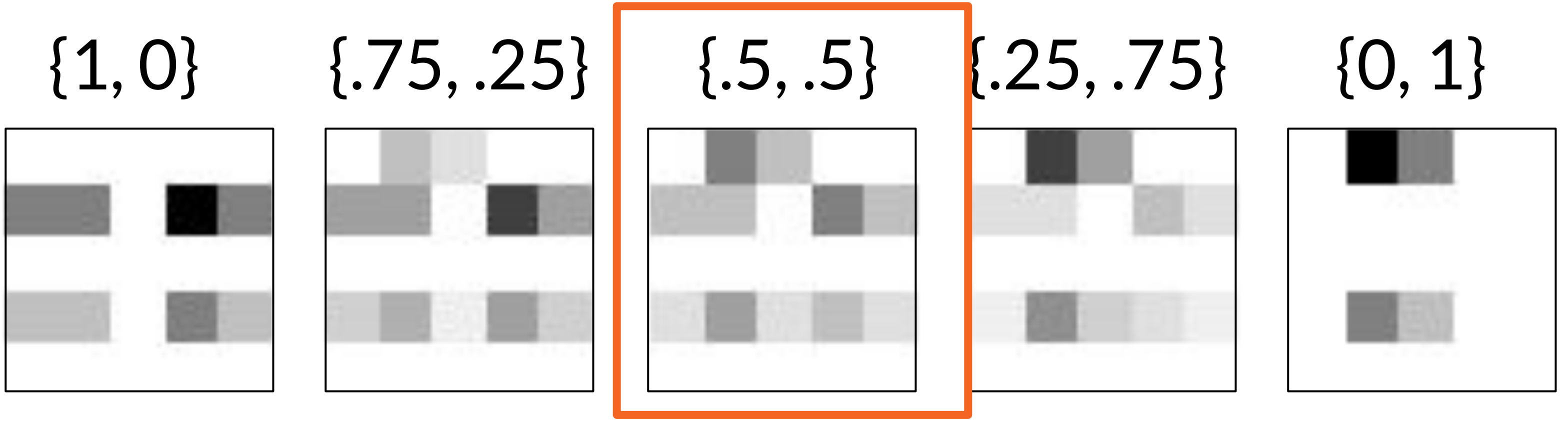
	4	2		
2	2		4	2
1	3	1	2	1



These inputs are unweighted



Linear combinations of matrices



Equal combination of two *component matrices*

Weights for components

Multiplying by the identity matrix in between won't change our final output

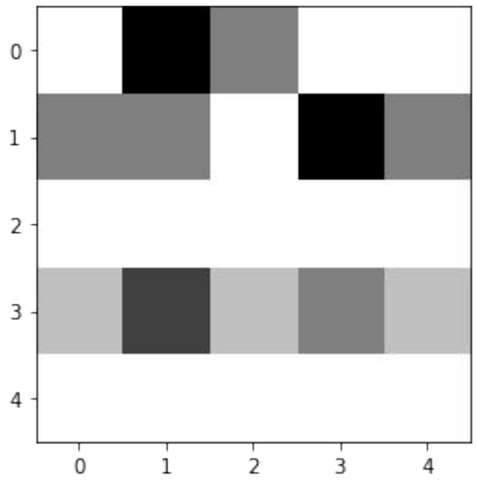
(the weights are the same for each of the two smaller matrices)

1	0
0	1

0	2	1	0	0
1	1	0	2	1

2	0
0	2
0	0
1	1
0	0

	4	2		
2	2		4	2
1	3	1	2	1



Weights for components

Matrix B (2x2)

1	0
0	1

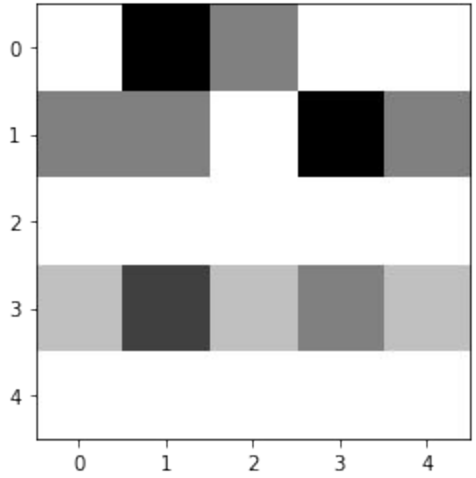
Matrix C (2x5)

0	2	1	0	0
1	1	0	2	1

Matrix A (5x2)

2	0
0	2
0	0
1	1
0	0

	4	2		
2	2		4	2
1	3	1	2	1



Weights for components

Matrix B (2x2)

1	0
0	1

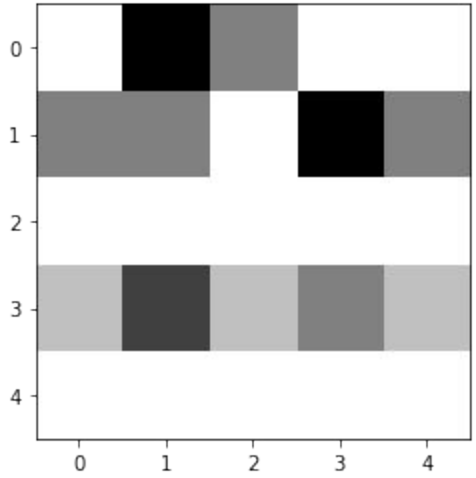
Matrix C (2x5)

0	2	1	0	0
1	1	0	2	1

Matrix A (5x2)

2	0
0	2
0	0
1	1
0	0

	4	2		
2	2		4	2
1	3	1	2	1



Multiplying ABC still gives us a 5x5 matrix!

Weights for components

Matrix B
(2x2)

1	0
0	1

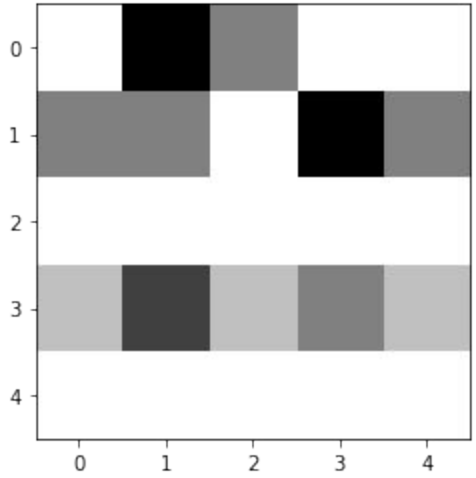
0	2	1	0	0
1	1	0	2	1

Matrix C
(2x5)

Matrix A
(5x2)

2	0
0	2
0	0
1	1
0	0

	4	2		
2	2		4	2
1	3	1	2	1



AB → shape 5x2
(AB)C → shape 5x5

Multiplying ABC still gives us a 5x5 matrix!

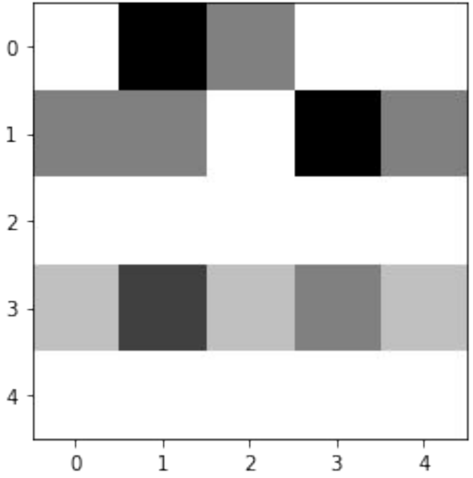
Adding matrices by multiplying vectors

1	0
0	1

0	2	1	0	0
1	1	0	2	1

2	0
0	2
0	0
1	1
0	0

	4	2		
2	2		4	2
1	3	1	2	1



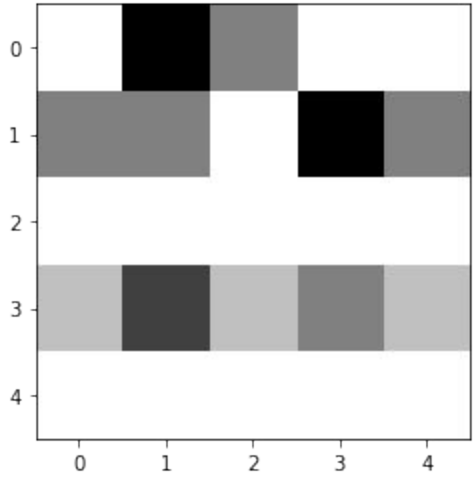
Adding matrices by multiplying vectors

1	0
0	1

0	2	1	0	0
1	1	0	2	1

2	0
0	2
0	0
1	1
0	0

	4	2		
2	2		4	2
1	3	1	2	1



Adding matrices by multiplying vectors

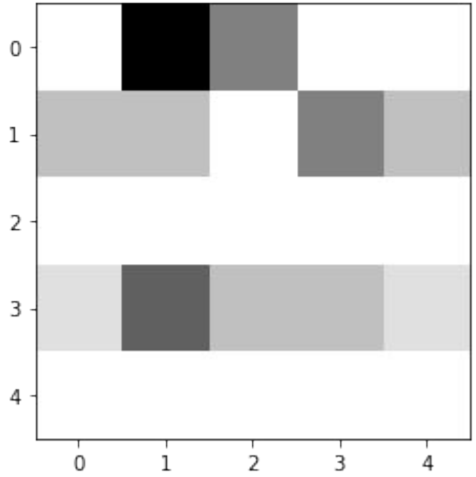
What if we make the first underlying matrix twice as important as the second underlying matrix?

2	0
0	1

0	2	1	0	0
1	1	0	2	1

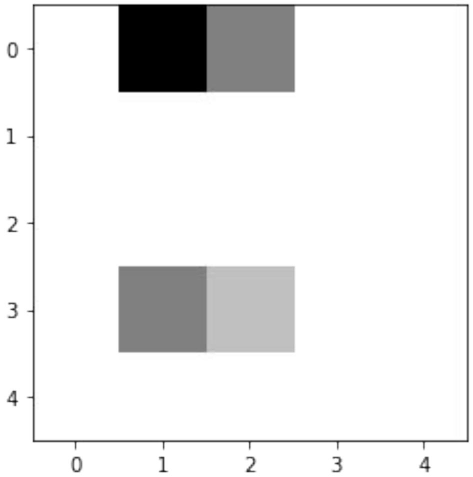
2	0
0	2
0	0
1	1
0	0

	8	4		
2	2		4	2
1	5	2	2	1

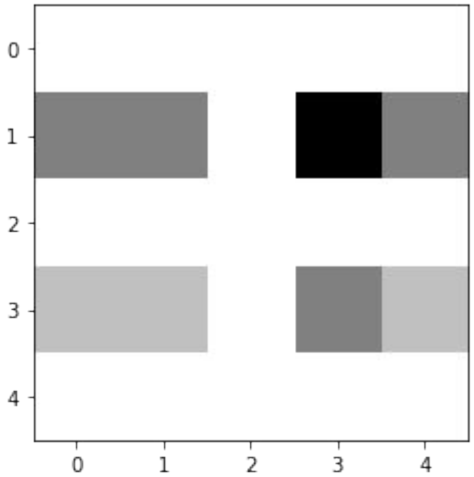


Adding matrices

2 *



+



= ?

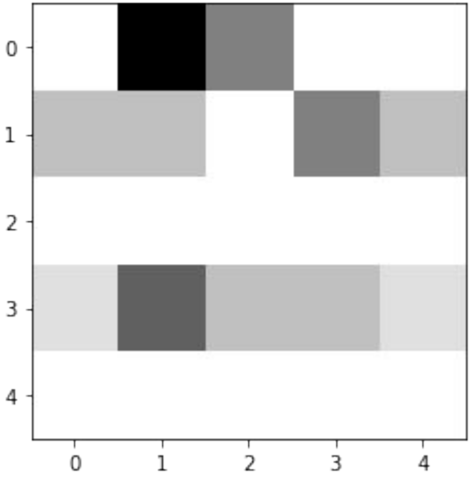
Adding matrices by multiplying vectors

2	0
0	1

0	2	1	0	0
1	1	0	2	1

2	0
0	2
0	0
1	1
0	0

	8	**4**		
2	2		4	2
1	**5**	**2**	2	1



Parts of a matrix factorization

2	0
0	1

0	2	1	0	0
1	1	0	2	1

2	0
0	2
0	0
1	1
0	0

Parts of a matrix factorization

2	0
0	1

0	2	1	0	0
1	1	0	2	1

2	0
0	2
0	0
1	1
0	0

Pairs of vectors that multiply to form a *component matrix*

Parts of a matrix factorization

2	0
0	1

0	2	1	0	0
1	1	0	2	1

2	0
0	2
0	0
1	1
0	0

We have two components, so the resulting matrix is "rank 2"

Parts of a matrix factorization

2	0
0	1

0	2	1	0	0
1	1	0	2	1

2	0
0	2
0	0
1	1
0	0

Diagonal matrix of weights for components

Posted by u/lidscrump 4 years ago

5.5k

Why in the world isn't there a Bob Ross palette yet?! He used the same 11 colors for every painting!

Discussion

https://www.reddit.com/r/MakeupAddiction/comments/apbbv6/why_in_the_world_isnt_there_a_bob_ross_palette/

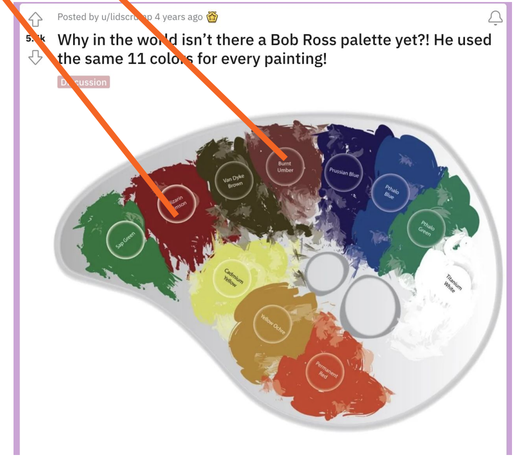
Parts of a matrix factorization

2	0
0	1

0	2	1	0	0
1	1	0	2	1

2	0
0	2
0	0
1	1
0	0

Components are like colors, component weights are how much of each color you are mixing



Singular Value Decomposition

1	0
0	1

0	2	1	0	0
1	1	0	2	1

2	0
0	2
0	0
1	1
0	0

We know how to construct a matrix from components and weights

Singular Value Decomposition

1	0
0	1

0	2	1	0	0
1	1	0	2	1

2	0
0	2
0	0
1	1
0	0

	4	2		
2	2		4	2
1	3	1	2	1

We know how to construct a matrix from components and weights

Singular Value Decomposition

	4	2		
2	2		4	2
1	3	1	2	1

SVD gives us components and weights *from* a matrix

Singular Value Decomposition

1	0
0	1

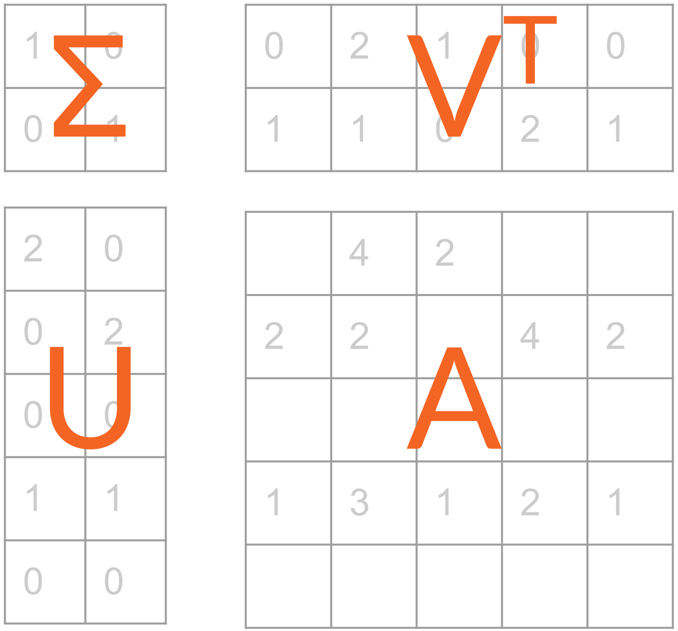
0	2	1	0	0
1	1	0	2	1

2	0
0	2
0	0
1	1
0	0

	4	2		
2	2		4	2
1	3	1	2	1

SVD gives us components and weights *from* a matrix

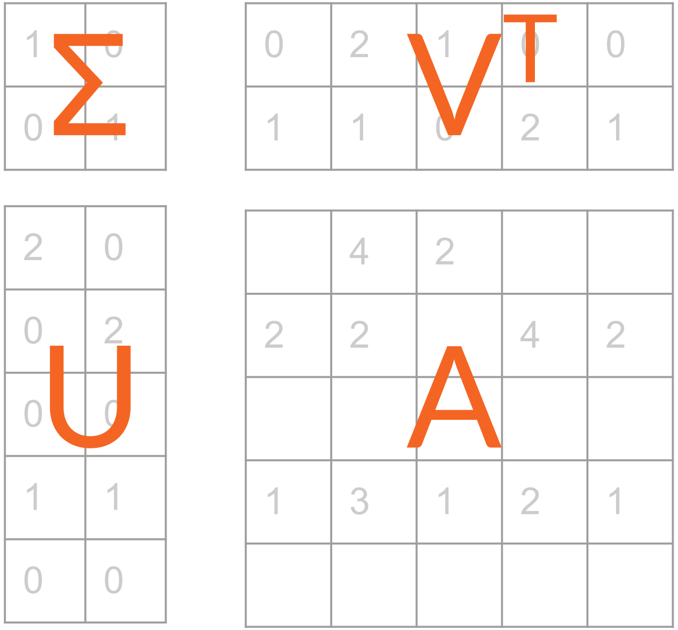
Singular Value Decomposition



SVD gives us components and weights *from* a matrix

Singular Value Decomposition

Σ is a matrix, not a summation!



SVD is a Decomposition into 3 matrices

It is **always** possible to decompose a *mxn* matrix A into the multiplication of three unique matrices: U, Σ, and V.

$$\boxed{A_{[m \times n]} = U_{[m \times r]}\, \Sigma_{[r \times r]}\, (V_{[n \times r]})^T}$$

- **A**: **Input data matrix**
  - *m* x *n* matrix (e.g., *m* documents, *n* terms)
- **U**: **Left singular vectors**
  - *m* x *r* matrix (*m* documents, *r* concepts)
- **Σ**: **Singular values**
  - *r* x *r* diagonal matrix (strength of each 'concept') (*r* : rank of the matrix **A**)
- **V**: **Right singular vectors**
  - *n* x *r* matrix (*n* terms, *r* concepts)

SVD is a Decomposition into 3 matrices

It is **always** possible to decompose a *mxn* matrix A into the multiplication of three unique matrices: U, Σ, and V.

$$\mathbf{A}_{[m \times n]} = \mathbf{U}_{[m \times r]} \, \Sigma_{[r \times r]} \, (\mathbf{V}_{[n \times r]})^T$$

- **A**: **Input data matrix**
  - *m* x *n* matrix (e.g., *m* documents, *n* terms)
- **U**: **Left singular vectors**
  - *m* x *r* matrix (*m* documents, *r* concepts)
- **Σ**: **Singular values**
  - *r* x *r* diagonal matrix (strength of each 'concept') (*r* : rank of the matrix **A**)
- **V**: **Right singular vectors**
  - *n* x *r* matrix (*n* terms, *r* concepts)

- ***U, V***: column orthonormal
  - $U^T U = I$; $V^T V = I$ (*I*: identity matrix)
  - (Columns are orthogonal unit vectors)
- Σ: diagonal
  - Entries (**singular values**) are positive, and sorted in decreasing order ($\sigma_1 \geq \sigma_2 \geq \ldots \geq 0$)

SVD in Python

```
U, S, Vt = np.linalg.svd(A)
```

Collaborative Filtering

What if there's a *method* that tackles the user-item recommendations using a *smaller representation*?

Singular Value Decomposition!!!

Identify similar movies from user interactions

Identify personality traits from surveys

Organize users from Stack Overflow tags

Compress images

Find similar documents from word counts

Impute missing values in a matrix

Singular Value Decomposition!!!

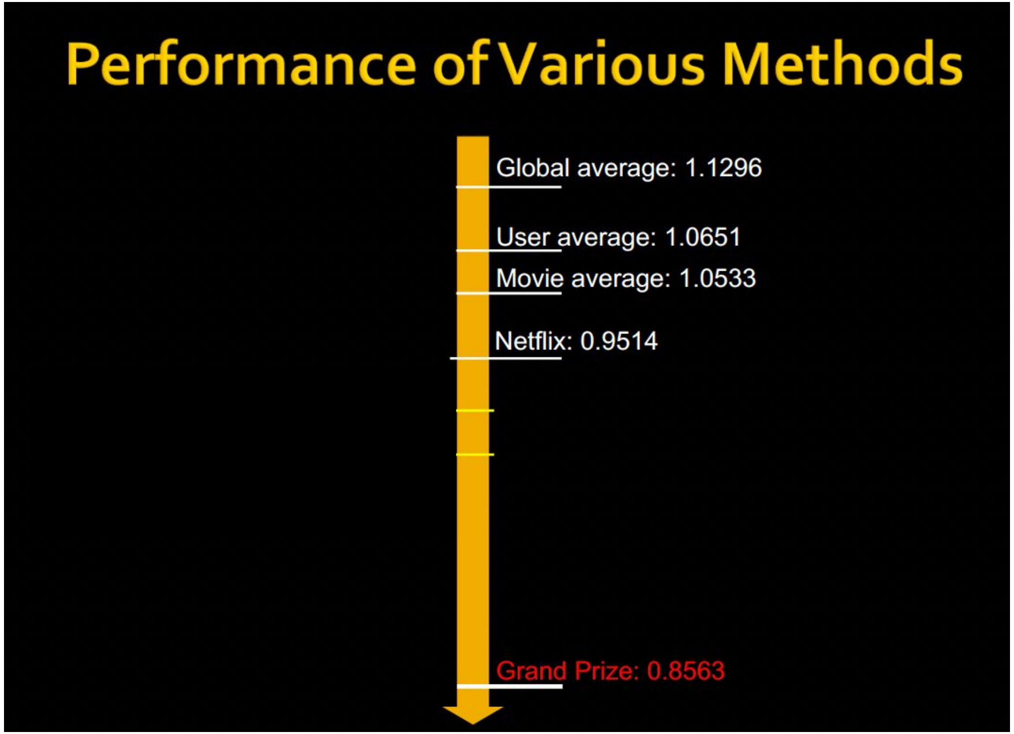
So many matrix factorizations used in the real world are just special cases of SVD!

https://twitter.com/daniela_witten/status/1400661490806390784

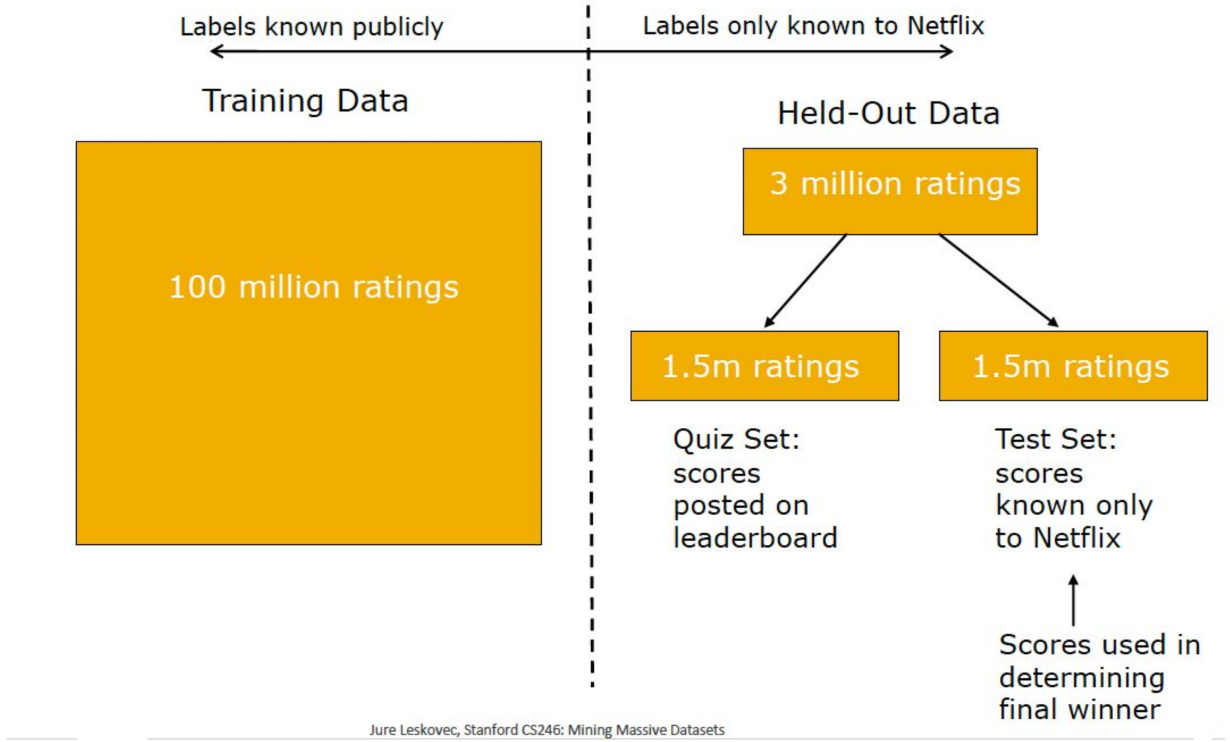
The Netflix Prize

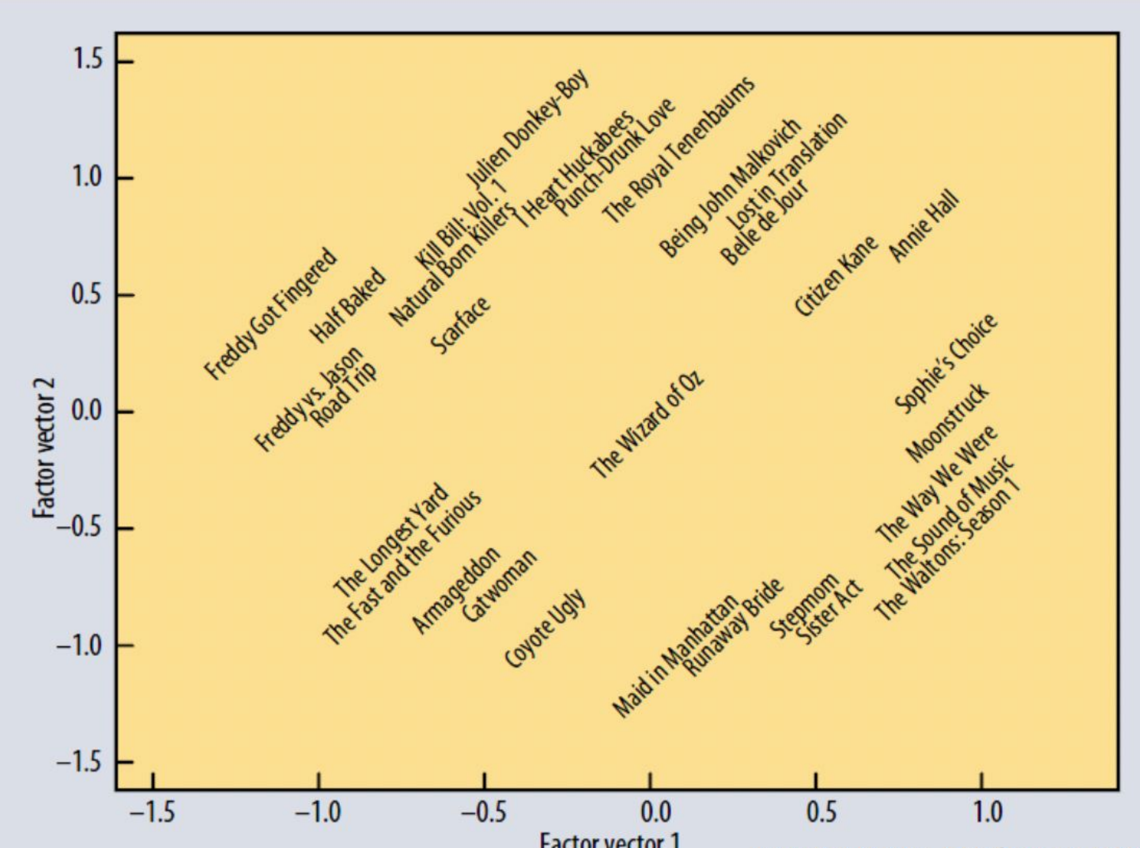
- Netflix announces a $1 million prize for 10% improvement on Netflix RMSE of 0.9514
    - Lower RMSE → better
- Training data publicly available:
    - 100 million ratings
    - 480,000 users
    - 17,770 movies
    - 6 years of data (2000-2005)

The Netflix Prize



The Netflix Prize

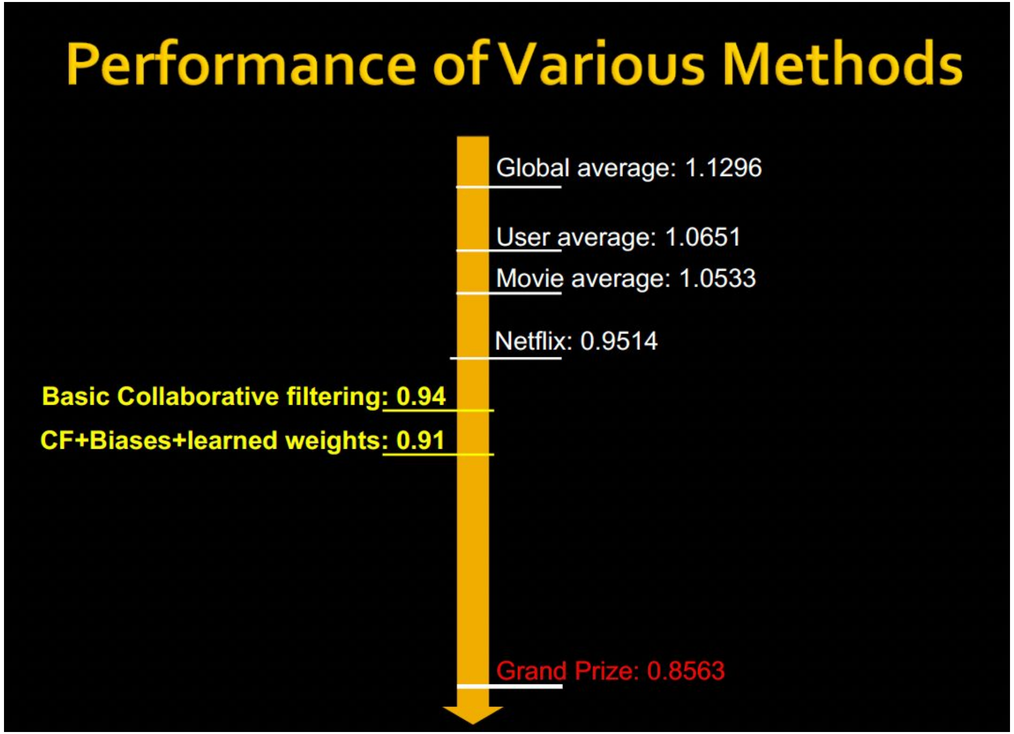




Koren, Bell, Volinksy, IEEE Computer, 2009

https://ieeexplore.ieee.org/document/5197422

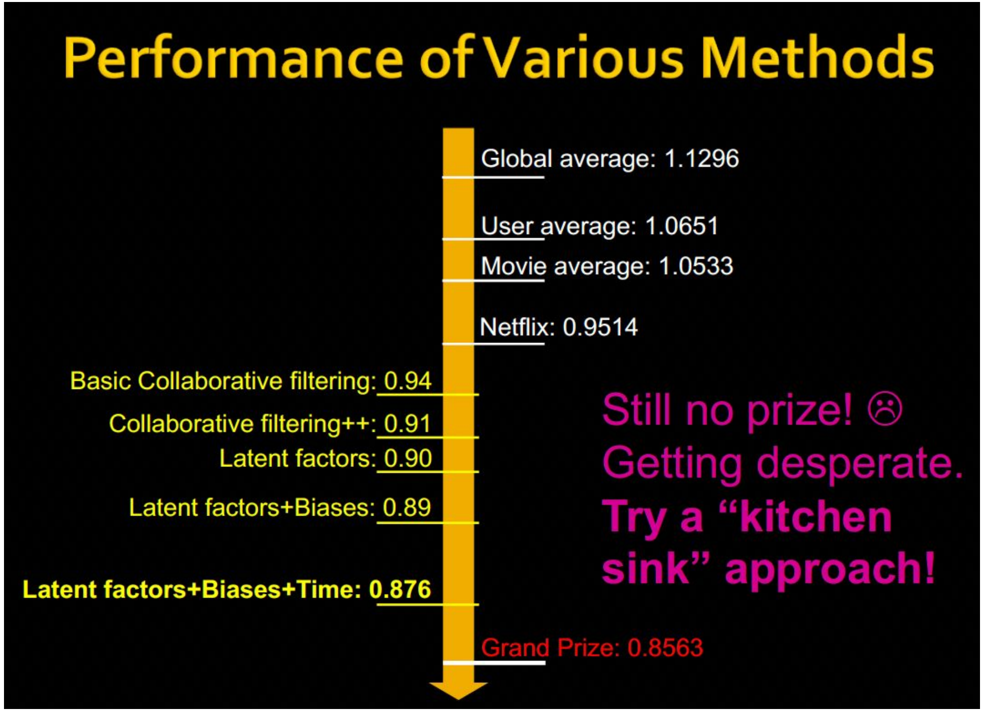
The Netflix Prize



The Netflix Prize

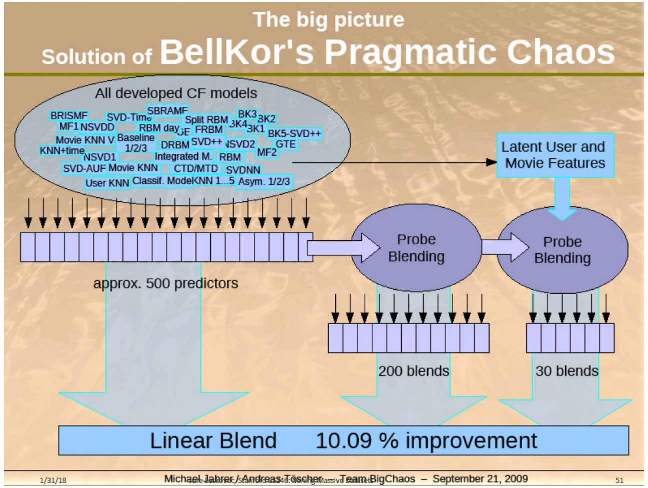
- No one gets close to the prize money for 2006-2008
- Potential temporal biases
  - Sudden rise in average movie ratings due to GUI improvements in Netflix
  - Things change over time!
    - Meaning of a 5 star rating
    - Preference for movies

The Netflix Prize



The Netflix Prize

- BellKor's kitchen sink method looks like this, beats the grand prize RMSE, and triggers a 30-day "last call" on June 26th, 2009



The Netflix Prize

- A bunch of other teams collude to combine their models (called an “ensembling method”)
    - E.g., ensembling can mean taking the average, mode, median, etc. of all the model predictions
- Ensemble method now outperforms Bellkor’s Pragmatic Chaos method

The Netflix Prize

- Teams were limited to 1 submission per day
- The final day of the prize call comes around
    - **40 minutes** before the deadline, BellKor submits a new model that does better than Ensemble
    - **20 minutes** before the deadline, Ensemble submits a new model too

The Netflix Prize

- BellKor and Ensemble submitted models that yielded a perfect tie
- BellKor won the $1 million because they submitted earlier!!

https://www.wired.com/2009/09/bellkors-pragmatic-chaos-wins-1-million-netflix-prize/