

# CSCI 4022 Fall 2021

## Collaborative Filtering

### Announcements and Reminders

- ▶ HW 7 due Monday
- ▶ Zach extended OH tonight (5p-6p)
- ▶ 3 Extra late days (now 5 total) for use over semester. Can't use more than 3 on one assignment, through.
- ▶ You don't *have* to have a partner for your project. It's definitely not necessary, and you **will** have to carry your weight in a group. You will be anonymously polled on group contributions, and it better be pretty equitable or grades will be commensurate.

## User-User recommendations

The big alternative to item-based recommendations is to instead recommend content based on similar other *users'* preferences, using *cosine* similarity. **Calculation:** normalize ratings by subtracting off the user's mean and still treating unknowns as 0. Now if we compute  $r_A \cdot r_B$ , we're multiplying their ratings-versus-personal-average against one another. Each multiplication is positive if both user's agree that the movie is **either** "above average" **or** "below average."

This is equivalent to the equivalent to Pearson correlation coefficient:

$$r_{XY} = \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^n (X_i - \bar{X})^2} \sqrt{\sum_{i=1}^n (Y_i - \bar{Y})^2}}$$

Sample Covariance

In other words, this is a measure of how well the scores for one user *correlate* to the scores of another user.

## Collaborative Filtering

Prediction based on the nearest (by ratings) neighbors is called *collaborative filtering*.

**Goal:** To predict user  $x$ 's unknown rating for item  $i$ .

Let  $r_x$  = vector of user  $x$ 's ratings

Let  $N$  = set of  $k$  users most similar to user  $x$  who have rated item  $i$

**Prediction:**

Option 1: Simple average of ratings in  $N$  for item  $i$ :

$$\hat{r}_{x,i} = \frac{1}{k} \sum_{y \in N} r_{y,i} \quad \text{average}$$

Option 2: Average of ratings in  $N$  for item  $i$ , *weighted* by user similarity to  $x$ :

$$\frac{1}{.9 + .7 + .85} (.9(3) + .7(4) + .85(4))$$

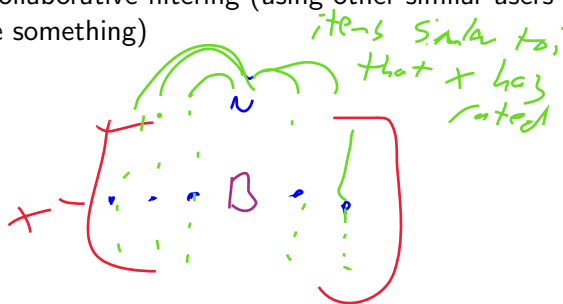
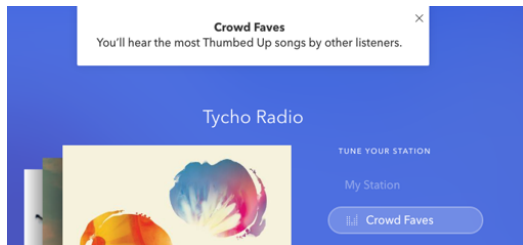
$$\hat{r}_{x,i} = \frac{1}{\sum_{y \in N} sim(x,y)} \sum_{y \in N} sim(x,y) r_{y,i}$$

- #1: .9
- #2: .7
- #3: .85



## Item-Item Filtering

So far, we have been talking about user-user collaborative filtering (using other similar users' ratings to predict whether a given user will like something)



There is an analogous view we could take: item-item collaborative filtering

For item  $i$ , find other similar items

Estimate rating for  $i$  based on ratings for similar items of that user

Can use all the same similarity and prediction functions as in our user-user model!

## Filtering Example

**Example:** Suppose the utility matrix for movies A-F and users 1-12 is given below. Ratings are between 1 (bad) and 5 (good), and blanks indicate unknown ratings. Use neighborhood of size  $|N| = 2$  and centered cosine similarity to estimate  $r_{A,5}$ .

	1	2	3	4	5	6	7	8	9	10	11	12
A	1		3			5			5		4	
B			5	4			4			2	1	3
C	2	4		1	2		3		4	3	5	
D		2	4		5			4			2	
E			4	3	4	2					2	5
F	1		3		3			2			4	

## Filtering Example

**Process** to estimate  $r_{A,5}$

1. **User-User** Check set of *columns* with entries for A. See how similar they are to user 5. Pick most similar 2, and *weighted average* their scores for 5.
2. **Item-Item** Check set of *rows* with entries for user 5. See how similar they are to movie A. Pick most similar 2, and *weighted average* their scores from user 5.

	1	2	3	4	5	6	7	8	9	10	11	12
A	1		3		X	5			5		4	
B			5	4			4			2	1	3
C	2	4		1	2		3		4	3	5	
D		2	4		5			4			2	
E			4	3	4	2					2	5
F	1		3		3			2			4	

## Filtering Example

**Process** to estimate  $r_{A,5}$

- User-User** Check set of *columns* with entries for A. See how similar they are to user 5.  
Pick most similar 2, and *weighted average* their scores for 5.

	1	2	3	4	5	6	7	8	9	10	11	12
A	1		3		X	5			5		4	
B			5	4			4			2	1	3
C	2	4		1	2		3		4	3	5	
D		2	4		5			4			2	
E			4	3	4	2					2	5
F	1		3		3			2			4	

The two most similar users are  $C$  and  $F$ , with  $s_{A,C} = 0.41$  and  $s_{A,F} = 0.59$ . Then the estimate is

$$\begin{aligned}
 \hat{r}_{A,5} &= \frac{\sum_N \text{sim}(A, N_A) r_{N_A,5}}{\sum_N \text{sim}(A, N_A)} \\
 &= \frac{1}{0.41 + 0.59} (0.41 \cdot 2 + 0.59 \cdot 3) \\
 &= 2.6.
 \end{aligned}$$

## Collaborative Filtering: Concerns

### Pros:

1. no feature selection necessary; works for any kind of data

### Cons:

1. Cold start – need enough users and items to find matches
2. Sparsity – user-ratings matrix is sparse, so hard to find users who have rated same items
3. First-rater problem – can't rec items with no ratings, or relatively few ratings
4. Popularity bias – tend to rec popular items since lots of users have rated them, and rated them highly (the “Harry Potter effect”)

So we have to choose between three possible options!



## Recs wrapup

1. Item/user profile based recommendations. **For each user**, model what they like.  
(predicted cosine similarity **or** predicted modeling, such as linear/ANOVA/logistic)
2. User-user collaborative filtering. **For each user**, see who behaves similarly.  
(k-nearest neighbors by rating)
3. Item-item collaborative filtering. **For each item**, see what behaves similarly.  
(k-nearest neighbors by rating)

Item-item vs user-user? In practice, item-item typically works better: items are simpler than users. Items belong to small genres, whereas users are diverse.

Choices may also depend on fundamental assumptions:

1. Do users buy items similar to other items in the catalogue, or are they exclusive/self-avoiding (things you only buy *once*).
2. Do we have more users or more items? Usually safer to use the ratings we have more of for prediction!

## Advanced Recs

Often, the best results are found by implementing two or more different recommender engines and combining predictions

1. Can combine using a linear model (e.g.  $\text{pred} = \text{wgt}_1 * \text{pred}_1 + \text{wgt}_2 * \text{pred}_2 + \dots$ )
2. Can combine a global baseline estimate with collaborative filtering

This can combine content-based methods to collaborative filtering!

1. Get item profiles to solve the new item problem
2. Get demographic information/average profiles to solve the new user problem

Rule of thumb and common practice: always combine some notion of overall mean ratings into user recommendations!

## Hybrid Recs

Suppose Janice has never rated any movie similar to The Sixth Sense, but we want to estimate her rating for it.

Define:

$\mu$  := overall mean movie rating

$b_x$  := rating deviation of user  $x$  = (average rating of user  $x$ ) -  $\mu$

$b_i$  = rating deviation of item  $i$  = (average rating of item  $i$ ) -  $\mu$

**Baseline estimate** of  $r_{x,i}$ , user  $x$ 's unknown rating of item  $i$ . Suppose...

Janice's mean movie rating overall: 3.7 stars

The Sixth Sense is 0.5 stars above average

Janice rates 0.2 stars below average

Baseline estimate is:

$$b_{x,i} = \mu + b_x + b_i$$

## Hybrid Recs

Suppose Janice has never rated any movie similar to The Sixth Sense, but we want to estimate her rating for it.

Define:

$\mu$  := overall mean movie rating

$b_x$  := rating deviation of user  $x$  = (average rating of user  $x$ ) -  $\mu$

$b_i$  = rating deviation of item  $i$  = (average rating of item  $i$ ) -  $\mu$

**Baseline estimate** of  $r_{x,i}$ , user  $x$ 's unknown rating of item  $i$ . Suppose...

Janice's mean movie rating overall: 3.7 stars

The Sixth Sense is 0.5 stars above average

Janice rates 0.2 stars below average

Baseline estimate is:

$$b_{x,i} = \mu + b_x + b_i = 3.7 - 0.2 + 0.5 = 4$$

## Hybrid Recs

As an aside, I find the reliance on means for *ratings* distasteful. Imagine Janice rated movie *on average* .8 points higher, and gets presented a movie that has a mean overall rating of 4.5/5. What do they rate it? 5.3/5 doesn't really make sense!

If you want to have mathematically consistent results, you have put a ceiling on this. Two options:

1. (Non-parametric): model each user in terms of their *quantiles*. If Janice is in the "top 90%" of highest raters, predict what this quantile would be for movie  $x$ .
2. (Parametric): Convert ratings to something where mean adjustments make sense, then convert back. Log-odds of rating/5 is a reasonable choice!

## Coding Best Practice

Note that we can seamlessly put this into our full model: *demean* each user, then *recenter* at the end.

**Baseline estimate:**  $b_{x,i} = \mu + b_x + b_i$

Combine this baseline estimate with collaborative filtering:

Old (CF-only):

$$\hat{r}_{x,i} = \frac{1}{\sum_{y \in N} \text{sim}(x, y)} \sum_{y \in N} \text{sim}(x, y) r_{y,i}$$

New (combined method):

$$\hat{r}_{x,i} = b_{x,i} \frac{1}{\sum_{y \in N; x} \text{sim}(x, y)} \sum_{y \in N; x} \text{sim}(x, y) (r_{y,i} - b_{x,j})$$

# Evaluations

For any system where we try to *predict* missing information, a common approach in machine learning is to create a **holdout set**.

**Process:** take a piece of the utility matrix (user-item-rating matrix) and withhold it from the rating estimation process as a test set,  $T$ . Then, compare predictions in  $T$  against the withheld known ratings  $T$ .

	movies									
users	1	3	4							
		3	5						5	5
			4	5					5	5
				3						
				3						
	2				2				2	
						5				
		2	1						1	
		3				3				
	1									

	movies									
users	1	3	4							
		3	5						5	5
			4	5					5	5
				3						
				3						
	2				?	?	?	?	?	?
					?	?	?	?	?	?
		2	1		?	?	?	?	?	?
		3			?	?	?	?	?	?
	1				?	?	?	?	?	?

We need a *distance* between the predictions and the true values as a score of *goodness*. One is:  
**root-mean-squared error (RMSE):**

$$RMSE = \sqrt{\frac{1}{|T|} \sum_{(x,i) \in T} (\underbrace{r_{x,i}}_{\text{true}} - \underbrace{\hat{r}_{x,i}}_{\text{predicted}})^2}$$

## Scoring Evaluations

Narrow focus on prediction accuracy can miss the point:

- ▶ Prediction diversity

Example: If a user likes Star Wars a lot, they might only be recommended Star Wars and very similar movies.

- ▶ Prediction context

Example: If a user buys a bunch of travel guides for an upcoming trip, then goes on the trip, no longer needs the travel recommendations.

- ▶ Order of predictions

Example: Should recommend users to watch Star Wars I: The Phantom Menace, before Star Wars II: Attack of the Clones, but Phantom Menace was terrible, so might not get recommended much.

In practice, we only care to predict high ratings anyway. So we care more about measuring how well our system does for matching high ratings



## Scoring Evaluations

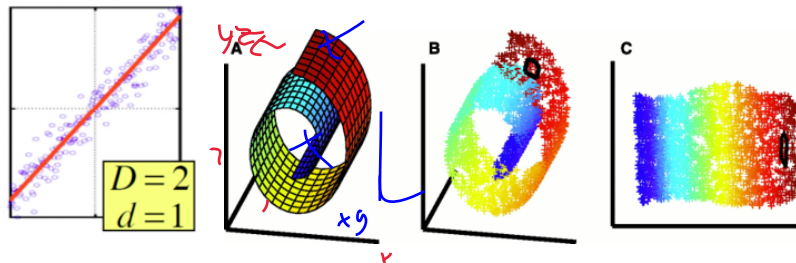
There are alternative to RMSE.

1. Other *metrics*: **precision at top 10%**, where we measure how often we correctly predict users' top 10 lists. Also **rank** correlation, e.g.  $\rho = 1 - \frac{6 \sum d_i^2}{n(n^2-1)}$  where  $d_i$  is difference in *rank* between predicted/actual.
2. Other *validation schema*. These include cross-validation like "Leave One Out (LOO)," where we fit the model many time. Remove a datum. Fit your model, then ask how well you predicted exactly that one datum without using it. Repeat over *all* datum. This is expensive but *very* appealing in theory compared to breaking the data into regular/holdout or training/prediction sets.

That concludes our discussion of recommendation systems, but we're not going far. The questions of predicting missing values or similarities on *matrix* structures is coming along. And we want some asides in linear algebra before we go any deeper.

## Dimension Reduction

The broad strokes of what linear algebra can give us is a more concise representation of data. Just like we use a number like  $\bar{X}$  to reduce a data set to one number, we often want to take entire data sets (matrices!) and *reduce* them into smaller representations.



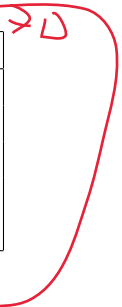
Basic assumption: We have data in  $D$  dimensions (features) but it actually (mostly) lies in a low or near a low  $d$ -dimensional subspace

The axes of this subspace are an effective, and often more efficient, representation of our data. Our goal: find those axes!

## Dimension Reduction

**Example** compressing/reducing dimensionality: Each data point in this matrix has 7 components (quantity purchased on each day of the week). So  $D = 7$  dimensions.

- ▶ Any row can be expressed as a scaled version of  $[1, 1, 1, 1, 1, 0, 0]$  or  $[0, 0, 0, 0, 0, 1, 1]$
- ▶ But this matrix is *really* 2-dimensional.
- ▶ How can we express this formally?



	Mon	Tue	Weds	Thurs	Fri	Sat	Sun
Dan's Can Co.	2	2	2	2	2	0	0
BriBear, Inc.	1	1	1	1	1	0	0
Cox Communications	5	5	5	5	5	0	0
Wong's Widgets	0	0	0	0	0	3	3
Mullen Manufacturing	0	0	0	0	0	2	2

## Dimension Reduction

**Definition:** The *rank* of a matrix is the number of linearly independent columns of  $A$ .

**Example:** Our matrix is rank of 2.

**Impact:** We can write any row of our matrix in terms of just 2 "basis" vectors:

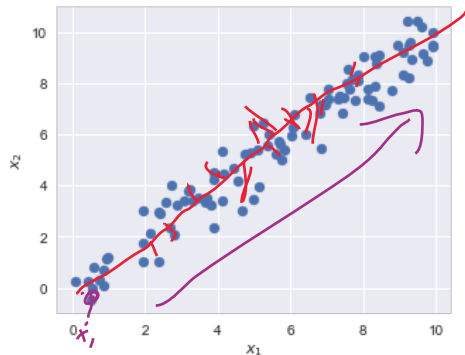
$$v_1 = [1, 1, 1, 1, 1, 0, 0] \text{ or } v_2 = [0, 0, 0, 0, 0, 1, 1]$$

	Mon	Tue	Weds	Thurs	Fri	Sat	Sun	Reduced Repr'n
Dan's Can Co.	2	2	2	2	2	0	0	[2,0]
BriBear, Inc.	1	1	1	1	1	0	0	[1,0]
Cox Communications	5	5	5	5	5	0	0	[5,0] + P, m, !
Wong's Widgets	0	0	0	0	0	3	3	[0,3]
Mullen Manufacturing	0	0	0	0	0	2	2	[0,2]

## Rank is Dimension

**Goal:** discover the "axis" of the data at the right.

Here,  $D = 2$ , but if we had to describe a data point using only one quantity, what would you use?



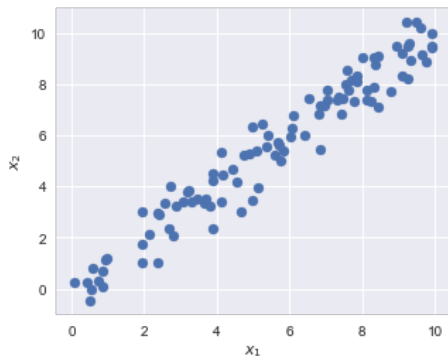
As we do this, we incur a small amount of error, because the data wasn't *actually* exactly on the line  $[1,1]$ . Dimension reduction requires a sense of **balance**: we want to use as few coordinates as possible, but with as little error as possible.

## Rank is Dimension

**Goal:** discover the "axis" of the data at the right.

Here,  $D = 2$ , but if we had to describe a data point using only one quantity, what would you use?

1. The important axis is the vector  $[1,1]$
2. We could instead just describe each points by how far along this axis it is!



As we do this, we incur a small amount of error, because the data wasn't *actually* exactly on the line  $[1,1]$ . Dimension reduction requires a sense of **balance**: we want to use as few coordinates as possible, but with as little error as possible.

# Why Reduce Dimension

1. Discover hidden correlations/topics/concepts

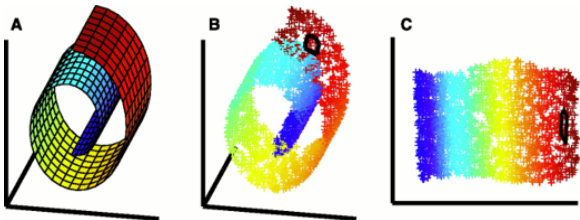
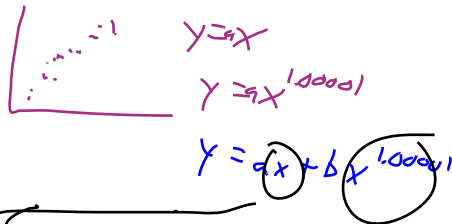
Example: words that occur together frequently

2. Remove redundant/noisy features – colinear

Example: some words are not useful for classifying/clustering documents

3. Interpretation and visualization is easier and more intuitive in fewer dimensions

4. Easier to store, process and analyze data in fewer dimensions



# Eigenvalues

A common step in dimension reduction is finding eigenvalues and eigenvectors of a symmetric  $n \times n$  matrix. So we need an algorithm for that.

**Question:** What does it mean for  $(\lambda, v)$  to be an eigenpair of a matrix  $A$ ?



## Eigenvalues

A common step in dimension reduction is finding eigenvalues and eigenvectors of a symmetric  $n \times n$  matrix. So we need an algorithm for that.

**Question:** What does it mean for  $(\lambda, v)$  to be an eigenpair of a matrix  $A$ ?

**Answer:** It means  $Av = \lambda v$  and  $v \neq 0$

1. So...  $(A - \lambda I)v = 0$
2. Since we insist on unit eigenvectors,  $|v| = 1$
3. Result:  $|A - \lambda I| = 0$  (linear algebra fact)

## Eigenvalues

A common step in dimension reduction is finding eigenvalues and eigenvectors of a symmetric  $n \times n$  matrix. So we need an algorithm for that.

**Question:** What does it mean for  $(\lambda, v)$  to be an eigenpair of a matrix  $A$ ?

**Answer:** It means  $Av = \lambda v$  and  $v \neq 0$

1. So...  $(A - \lambda I)v = 0$
2. Since we insist on unit eigenvectors,  $|v| = 1$
3. Result:  $|A - \lambda I| = 0$  (linear algebra fact)

**(pen-and-paper) Algorithm:** So solving for eigenvalues amounts to writing down the determinant  $|A - \lambda I|$  (a polynomial) and solving for its roots.

Then, we can set up and solve the linear system  $Av = \lambda v$  (with the restriction that  $|v| = 1$ ) to find the associated eigenvector.

## Exact Eigenvalues

**Example:** Find the eigenvalue/eigenvector pairs for  $A = \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix}$

## Exact Eigenvalues

**Example:** Find the eigenvalue/eigenvector pairs for  $A = \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix}$

**Step 1:** Characteristic Polynomial.

$$\begin{aligned} |A - \lambda I| &= \left| \begin{bmatrix} 3 - \lambda & 2 \\ 2 & 6 - \lambda \end{bmatrix} \right| = \det \begin{bmatrix} 3 - \lambda & 2 \\ 2 & 6 - \lambda \end{bmatrix} \\ &= (3 - \lambda)(6 - \lambda) - 2 \cdot 2 \implies \\ 0 &= \lambda^2 - 9\lambda + 14 \implies \lambda = 2, 7 \end{aligned}$$

## Exact Eigenvalues

**Example:** Find the eigenvalue/eigenvector pairs for  $A = \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix}$

**Step 2:** Eigenvectors. Using  $\lambda_1 = 7$ , we find  $v_2$  via  $Av_2 = \lambda_2 v$ . Suppose  $v_2 = [x, y]^T$ .

$$\begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 7 \begin{bmatrix} x \\ y \end{bmatrix} \implies \begin{bmatrix} 3x + 2y \\ 2x + 6y \end{bmatrix} = \begin{bmatrix} 7x \\ 7y \end{bmatrix}$$

**Both** rows of this system suggest that  $y = 2x$ , so we combine with  $x^2 + y^2 = 1$  to get

$$v_2 = \begin{bmatrix} 1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix}$$

## Exact Eigenvalues

**Example:** Find the eigenvalue/eigenvector pairs for  $A = \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix}$

**Final Result:** The matrix  $A$  has eigenpairs of  $(\lambda, \mathbf{v})$  of  $(7, [1/\sqrt{5}, 2/\sqrt{5}]^T)$  and  $(2, [2/\sqrt{5}, -1/\sqrt{5}]^T)$

## Exact Eigenvalues

In practice, this is pretty awful for huge matrices, for a few reasons!

1. We often only care about *largest* eigenvalues, just like for PageRank or H/A.
2. But solving a characteristic polynomial finds *all* the eigenvalues, and it's hard to guarantee we find the largest.
3. Determinants are *very* computationally expensive

On the other hand... *we need more than just one eigenvalue*. We want **some** of the largest, but maybe not the full eigenspace. We can use *power iteration*, but **generalized power iteration** to iteratively and sequentially find eigenvalues from largest-to-smallest

## Computational Eigenvalues

**Idea:** Find the largest eigenvalue via power iteration. Then somehow *remove* it from the matrix, so now the *second highest* has become the highest. Repeat power iteration!

**Recall: Power Iteration**



## Computational Eigenvalues

**Idea:** Find the largest eigenvalue via power iteration. Then somehow *remove* it from the matrix, so now the *second highest* has become the highest. Repeat power iteration!

### Recall: Power Iteration

Let  $A$  be a symmetric  $n \times n$  matrix, whose eigenstuff we want to find. Start with some  $x_0 \neq 0$ , then

1. Iterate until convergence:  $\|x_k - x_{k+1}\|_F < tol$
2. Then our final  $x$  is the *principal* eigenvector of  $A$ ,  $x_1$ , and we can solve for the associated eigenvalue  $\lambda_1$  via:

$$Ax = \lambda x \implies x^T Ax = x^T \lambda x \implies \lambda = x^T Ax$$

*Note:* the matrix/vector norm used there is the Frobenius norm:

$$\|A_F\| = \sqrt{\sum_{i,j} A_{i,j}^2}$$

# Computational Eigenvalues

**How do we remove an eigenvalue to set up finding the second-biggest?**

**Process:** Set

$$A_2 = A - \lambda_1 x_1 x_1^T$$

## Computational Eigenvalues

**How do we remove an eigenvalue to set up finding the second-biggest?**

**Process:** Set

$$A_2 = A - \lambda_1 \underbrace{x_1 x_1^T}_{\text{matrix with (i,j) component } x_i x_j}$$

...and then do power method!

1. Iterate exactly the same way that led you to  $x_1$  to find  $x_2$ , the eigenvector associated with the second-largest eigenvalue.
2. Solve for the second eigenvalue the same way too, as:  $\lambda_2 = x_2^T A x_2$
3. and continue until you found them all (or however many you wanted), updating the matrix as

$$A_{k+1} = A_k - \lambda_k x_k x_k^T$$

## Computational Eigenvalues

**How do we remove an eigenvalue to set up finding the second-biggest?**

**Process:** Set

$$A_2 = A - \lambda_1 \underbrace{x_1 x_1^T}_{\text{matrix with (i,j) component } x_i x_j}$$

...and then do power method!

1. Iterate exactly the same way that led you to  $x_1$  to find  $x_2$ , the eigenvector associated with the second-largest eigenvalue.
2. Solve for the second eigenvalue the same way too, as:  $\lambda_2 = x_2^T A x_2$
3. and continue until you found them all (or however many you wanted), updating the matrix as

$$A_{k+1} = A_k - \lambda_k x_k x_k^T$$

## Theory for Power Iteration

Can we convince ourselves that  $A_2 = A - \lambda_1 x_1 x_1^T$  “removes” the eigenvector  $x_1$  from  $A$ ?  
What does that even mean?

**Proof:** Suppose  $(\lambda, x)$  is an eigenpair of matrix  $A$ , and  $x \neq x_1$  (the principal eigenvector).

**Claim:**  $(\lambda, x)$  is an eigenpair of  $A_2$ .

**Proof:**

**Claim:**  $x_1$  is also an eigenvector of  $A_2$ , but its corresponding eigenvalue is 0.

**Proof:**

## Theory for Power Iteration

Can we convince ourselves that  $A_2 = A - \lambda_1 x_1 x_1^T$  “removes” the eigenvector  $x_1$  from  $A$ ?  
What does that even mean?

**Proof:** Suppose  $(\lambda, x)$  is an eigenpair of matrix  $A$ , and  $x \neq x_1$  (the principal eigenvector).

**Claim:**  $(\lambda, x)$  is an eigenpair of  $A_2$ .

**Proof:**  $A_2 x = (A - \lambda_1 x_1 x_1^T) x = Ax - \lambda_1 x_1 x_1^T x = \lambda x - \lambda_1 (0) = \lambda x \checkmark$

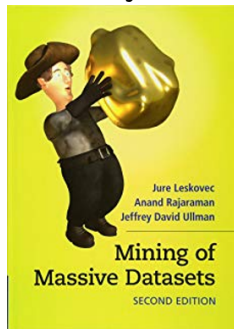
**Claim:**  $x_1$  is also an eigenvector of  $A_2$ , but its corresponding eigenvalue is 0.

**Proof:**  $A_2 x_1 = (A - \lambda_1 x_1 x_1^T) x_1 = Ax_1 - \lambda_1 x_1 x_1^T x_1 = \lambda_1 x_1 - \lambda_1 x_1 (1) = 0 = 0x_1 \checkmark$

## Acknowledgments

Next time: Using eigenstuff to reduce matrices!

Some material is adapted/adopted from Mining of Massive Data Sets, by Jure Leskovec, Anand Rajaraman, Jeff Ullman (Stanford University) <http://www.mmds.org>



Special thanks to Tony Wong for sharing his original adaptation and adoption of slide material.