THE UNIVERSITY OF TEXAS AT AUSTIN
Department of Statistics and Data Sciences
College of Natural Sciences

SDS 385: Stat Models for Big Data

Lecture 1: Introduction

Purnamrita Sarkar
Department of Statistics and Data Science
The University of Texas at Austin

https://psarkar.github.io/teaching

Managerial Stuff

- Instructor- Purnamrita Sarkar
- Course material and homeworks will be posted under `https://psarkar.github.io/sds385.html`
- Office hours: TBA. Welch Hall 5.228B
- TA: Rimli Sengupta
- Grading -
  - 3-4 homeworks (30%), weekly paper presentation(40%), Take home exam (30%)
  - I will make a list of papers for presentation within a week. Please start signing up. You need to do the presentations in groups of 2.
- Homeworks:
  - Homeworks are due (every 2-3 weeks) before midnight on Sundays via Canvas.
  - All homeworks need to be typed up in latex.

Managerial Stuff

If you are not feeling well and need to stay home,

- I will make sure whether we do things online or in person, the course materials is always recorded.

Managerial Stuff

- Weekly presentation format:
  - Wednesdays we will have presentations from a group of 2/3 students for 40-50 minutes, where each should present a significant part of the paper. The presentation is worth 15% of the grade.
  - After the presentation, there will be a breakout session discussing the paper. This is 5% of the grade
  - By midnight each Wednesday, all of you should submit a one page review of the paper in question. The best way to do this would be to **read the paper before the class and start working on the review**. This will be 20% of the grade.
  - The review should be single-spaced and one page. Very similar to a NeurIPS review, I will look for three parts:
    - Summarize the paper,
    - Discuss the contribution (e.g., theoretical, methodological, algorithmic, empirical contributions). For each contribution, briefly state the level of significance (i.e., how much impact will this work have on researchers and practitioners in the future?). If you cannot think of three things, please explain why. Not all good papers will have three contributions),
    - Discuss the strengths and weaknesses.
    - (Don't do:) say vague things, for example, "the paper makes an important contribution and is a very important piece of work because it has very nice algorithms"

Prerequisites

- Linear algebra
- Undergrad probability
- Calculus
- And be comfortable with programming
    - You can use any programming language of your choice
    - R/Matlab/Python

What is big data?

- When algorithms with complexity $O(n)$ becomes "intractable".
  - What is $O(n)$? We will do that in a sec.
- Take the simple method of matrix vector multiplication.
  - You have a $n \times n$ matrix X, and a length n vector y.
  - You want to compute the matrix vector product.

What is big data?

- When algorithms with complexity $O(n)$ becomes "intractable".
  - What is $O(n)$? We will do that in a sec.
- Take the simple method of matrix vector multiplication.
  - You have a $n \times n$ matrix X, and a length n vector y.
  - You want to compute the matrix vector product.
  - How long does that take? $O(n^2)$ time.
  - Say you have $n = 1M$. $O(n^2)$ is pretty bad!
  - Even if you have $n = 10K$ and the matrix vector product needs to be computed often in the course of your algorithm, its bad!

What is big data?

- Even storage can be an issue.
- Say $n = 1M$ and you need to compute nearest neighbors for KNN classification.

What is big data?

- Even storage can be an issue.
- Say $n = 1M$ and you need to compute nearest neighbors for KNN classification.
- If you store the entire $n \times n$ distance matrix that requires $O(n^2)$ storage. This may be too much.
- May need on-the-fly distance computations.
- Sometimes in Google/Facebook the data is so enormous that in order to access neighbors of neighbors of an webpage or entity may require communicating with another machine.
- Typically we want to store as much data as possible in RAM, since disk seeks are more time consuming.

Where else does the need for large scale learning come up

- Matrix inversion shows up in many estimation problems like regression.

Where else does the need for large scale learning come up

- Matrix inversion shows up in many estimation problems like regression.
  - Takes $O(n^3)$ time.
  - How about linear solvers instead?

Where else does the need for large scale learning come up

- Matrix inversion shows up in many estimation problems like regression.
  - Takes $O(n^3)$ time.
  - How about linear solvers instead?
- Nearest neighbor computation shows up in KNN classification.

Where else does the need for large scale learning come up

- Matrix inversion shows up in many estimation problems like regression.
  - Takes $O(n^3)$ time.
  - How about linear solvers instead?
- Nearest neighbor computation shows up in KNN classification.
  - Computing distances to n items $O(np)$ time.

Where else does the need for large scale learning come up

- Matrix inversion shows up in many estimation problems like regression.
  - Takes $O(n^3)$ time.
  - How about linear solvers instead?
- Nearest neighbor computation shows up in KNN classification.
  - Computing distances to n items $O(np)$ time.
  - Sorting these $n-1$ numbers take about $n \log n$ time.
  - How about approximate nearest neighbor methods and distance preserving low dimensional transformations?

This class

How do you deal with Big data? We come up with ways of representing the data in a more concise fashion.

- Look at the data one point at a time.
  - Online algorithms
  - Stochastic gradient descent, and various iterative optimization methods

This class

How do you deal with Big data? We come up with ways of representing the data in a more concise fashion.

- Look at the data one point at a time.
  - Online algorithms
  - Stochastic gradient descent, and various iterative optimization methods
- Divide and conquer: distributed algorithms
- Sampling based algorithms/ randomized algorithms
  - Approximating matrix multiplication by row/column sampling
  - Johnson Lindenstrauss Lemma and related fast projection methods
  - Hashing and sketching (also extremely useful for nearest neighbor calculation)

This class

How do you deal with Big data? We come up with ways of representing the data in a more concise fashion.

- Look at the data one point at a time.
  - Online algorithms
  - Stochastic gradient descent, and various iterative optimization methods
- Divide and conquer: distributed algorithms
- Sampling based algorithms/ randomized algorithms
  - Approximating matrix multiplication by row/column sampling
  - Johnson Lindenstrauss Lemma and related fast projection methods
  - Hashing and sketching (also extremely useful for nearest neighbor calculation)
- Sampling based methods for uncertainty estimation
  - Bootstrap
  - Subsampling

This class

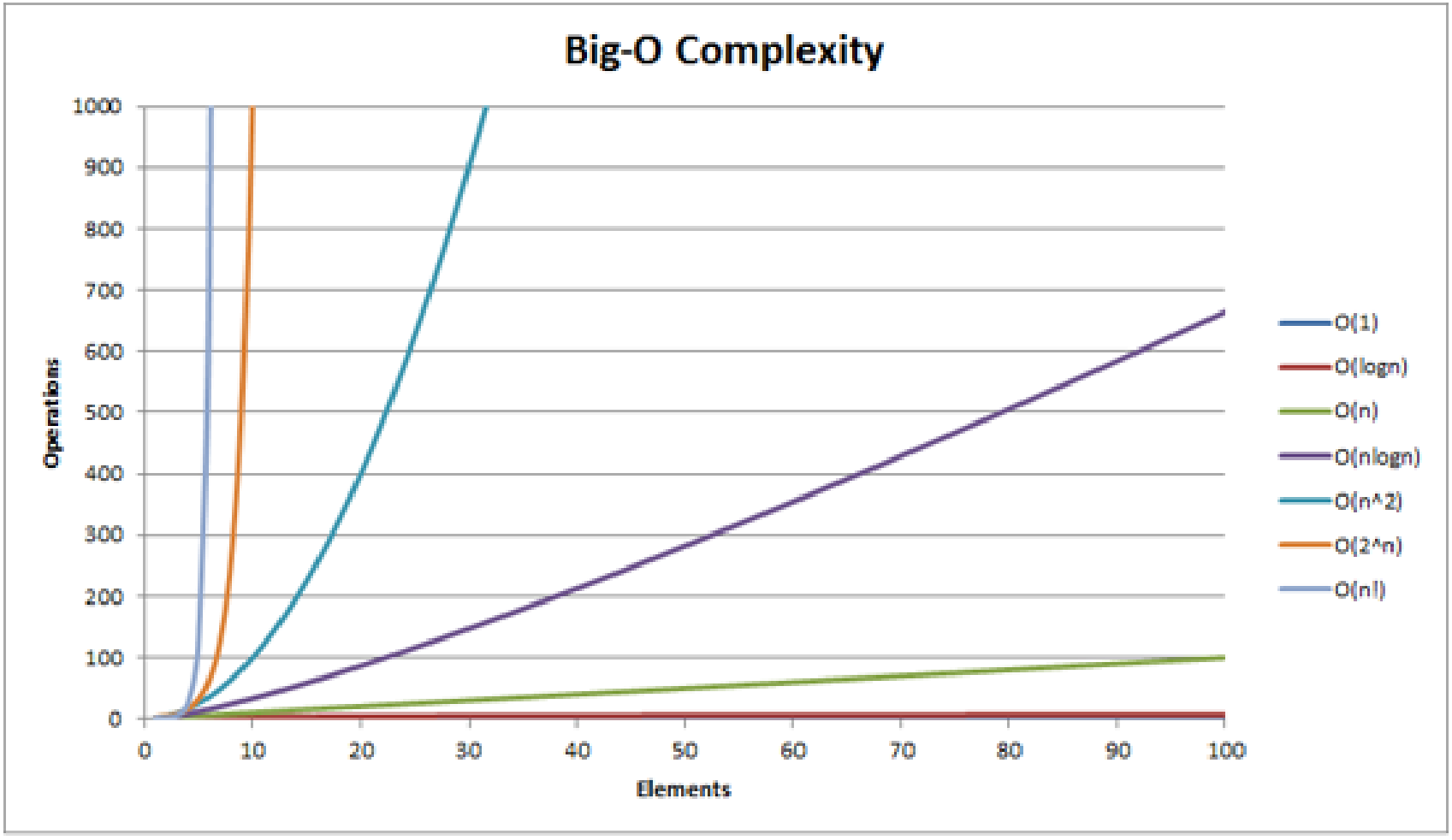
How do you deal with Big data? We come up with ways of representing the data in a more concise fashion.

- Look at the data one point at a time.
  - Online algorithms
  - Stochastic gradient descent, and various iterative optimization methods
- Divide and conquer: distributed algorithms
- Sampling based algorithms/ randomized algorithms
  - Approximating matrix multiplication by row/column sampling
  - Johnson Lindenstrauss Lemma and related fast projection methods
  - Hashing and sketching (also extremely useful for nearest neighbor calculation)
- Sampling based methods for uncertainty estimation
  - Bootstrap
  - Subsampling
- Ranking methods
  - Random walk based methods like Pagerank
  - Semisupervised learning
  - Manifold regularization

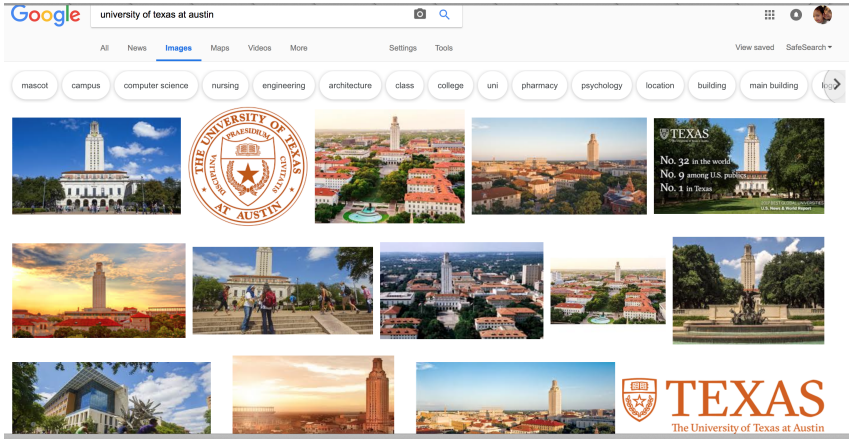
Order notations

- We define $f(x) = O(g(x))$ as $x \to \infty$ iff $\exists M > 0$ and $x_0 \in \Re$, such that $|f(x)| \leq M|g(x)|\ \forall x \geq x_0$.
- Typically for this class, x will be n, the number of data points in your dataset, and we will use the order notation when $n \to \infty$.
- For any task at hand, if you have an algorithm that runs in say $g(n)$ time, you say that the computation time is $O(g(n))$, since there could be a faster algorithm than the one you have.
- There are also o Ω and ω notations, which we wouldn't need very much for this class.

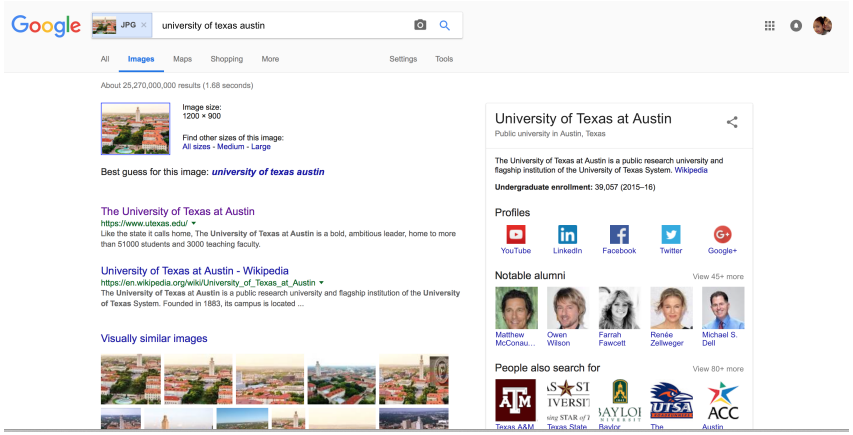
Order notations



Big data examples



(A)



(B)

Figure 1: (A) Image search (B) Reverse image search

- There are over a Billion images on the web.
- Each image is a high dimensional object
- There are possibly millions of categories of these images
- Instagram sees about 40M photos uploaded per day; its users give 8,500 likes and 1,000 comments per second.
- Whatsapp users share about 5B photos and 1B videos per day

Social Media

Figure 2: Fake news

- Whatsapp sees about 55B messages each day.
- Twitter sees 350K tweets sent per minute.
- How do we detect fake news when it is being forwarded by one user to another?
- First challenge would be how to classify, since there are not too many labeled news items. Even if you could figure out a way, how will you scale it to this humongous downpour of data?

Big data: the good, the bad

- Pros:
  - Most statistical consistency guarantees say that if number of datapoints go to infinity, then one can learn the parameters with high precision.
  - So, arguably, if I give you a billion data points, then you can just run a simple method and get a pretty good accuracy.

Big data: the good, the bad

- Pros:
  - Most statistical consistency guarantees say that if number of datapoints go to infinity, then one can learn the parameters with high precision.
  - So, arguably, if I give you a billion data points, then you can just run a simple method and get a pretty good accuracy.
- Cons:
  - How do you scale an existing algorithm to handle Billions of data points, or data points arriving in a stream?
  - As the number of datapoints grow, it is natural to assume that you see more and more categories, and so classification and clustering gets harder and harder!

Big data: the ugly[1]

- As the number of datapoints grow, the number of features grow too. So now we are looking at high dimensional data.

[1] Michael I. Jordan's "Why Big Data Could be a Big Fail" at spectrum.ieee.org

Big data: the ugly[1]

- As the number of datapoints grow, the number of features grow too. So now we are looking at high dimensional data.
- So one can get an outpouring of false positives.
- Take Mike's example: "if you live in Beijing, and you ride bike to work, and you work in a certain job, and are a certain age – what's the probability you will have a certain disease or you will like my advertisement?"

[1] Michael I. Jordan's "Why Big Data Could be a Big Fail" at spectrum.ieee.org

Big data: the ugly[1]

- As the number of datapoints grow, the number of features grow too. So now we are looking at high dimensional data.
- So one can get an outpouring of false positives.
- Take Mike's example: "if you live in Beijing, and you ride bike to work, and you work in a certain job, and are a certain age – what's the probability you will have a certain disease or you will like my advertisement?"
- Because of the many attributes, one may be able to predict any outcome with zero error, just by chance.
- How to fix this? Well how about error bars? They will tell you whether the outcome is actually surprising or not. Most learning algorithms do not actually compute these.

[1] Michael I. Jordan's "Why Big Data Could be a Big Fail" at spectrum.ieee.org

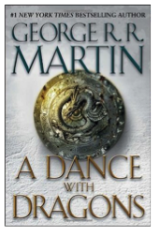
Big data: the ugly[2]

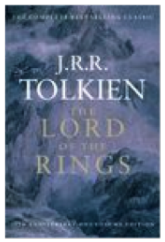
- And then the gem: "... if people use data and inferences they can make with the data without any concern about error bars, about heterogeneity, about noisy data, about the sampling pattern, about all the kinds of things that you have to be serious about if you are an engineer and a statistician – then you will make lots of predictions, and there's a good chance that you will occasionally solve some real interesting problems. But you will occasionally have some disastrously bad decisions. And you won't know the difference a priori. You will just produce these outputs and hope for the best."

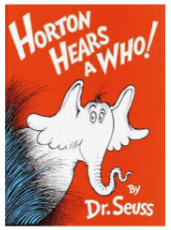
[2] Michael I. Jordan's "Why Big Data Could be a Big Fail" at spectrum.ieee.org

Examples: Binary classification

1. Given n datapoints $X_1, \ldots, X_n \in \mathbb{R}^d$, and their labels $Y_1, \ldots, Y_n \in \{0, 1\}$, output a decision function $f : \mathbb{R}^d \to \{0, 1\}$.







bell jar sylvia plath queer sultry summer

Y=0

george martin dance dragons warg valyria thistle

Y=0

where wild things maurice sendak max journey imaginative

Y=1

lord rings tolkien fellowship frodo mordor pippin baggins

Y=0

horton hears who seuss elephant timeless classic comical

Y=1

giving tree shel silverstein poignant picture book unforgettable

Y=?

Methods-lets try kNN

- How do you create features for a document?

Methods-lets try kNN

- How do you create features for a document?
- Bag of words representation: each word is a feature and the value f_{ij} is the term frequency of word i in document j.

Methods-lets try kNN

- How do you create features for a document?
- Bag of words representation: each word is a feature and the value f_{ij} is the term frequency of word i in document j.
- Often one uses tf-idf which is $f_{ij} \log N/n_i$ where N is the total number of documents and n_i is the number of documents where word i appears.

Methods-lets try kNN

- How do you create features for a document?
- Bag of words representation: each word is a feature and the value f_{ij} is the term frequency of word i in document j.
- Often one uses tf-idf which is $f_{ij} \log N/n_i$ where N is the total number of documents and n_i is the number of documents where word i appears.
- This gives less weight to words that are very frequent across all documents.
- Say X_i is in $\mathbb{R}^V$ where V is the size of vocabulary.
- How will you do k-NN classification?

Methods-lets try kNN

- Compute distance to all other datapoints.
- Each distance computation takes

Methods-lets try kNN

- Compute distance to all other datapoints.
- Each distance computation takes
  - $O(V)$ time.
- N distance computations take
  - $O(NV)$ time.
- k-nearest neighbors take $O(N \log N)$ time.
  - $O(N \log N)$ time.
- So, all in all $O(N \log N + NV)$ time.
- Goodreads has about $2B$ books, vocabulary size could be 10^4.
- If each multiplication or addition took 10^{-12} seconds, then the nearest neighbor computation would take about 30 seconds.

How to fix this- mock run

- Use **sparsity**, even though the vocabulary size is huge, not all words are used in every document, i.e. the input data vectors are sparse.
- Try **dimensionality reduction**: use SVD or random projections or feature selection
- Try **cleverer ways of computing nearest neighbors**, so that you only compute distances to "potential" nearest neighbors, not all points – Locality sensitive hashing, KD-trees, cover trees

Multiclass classification

- Given n datapoints $X_1, \ldots, X_n \in \mathbb{R}^d$, and their labels $Y_1, \ldots, Y_n \in \{0, 1, \ldots, K-1\}$, output a decision function $f : \mathbb{R}^d \to \{0, 1, \ldots, K-1\}$.
- In document classification, you would have multiple categories, these categories can also have hierarchical structure.

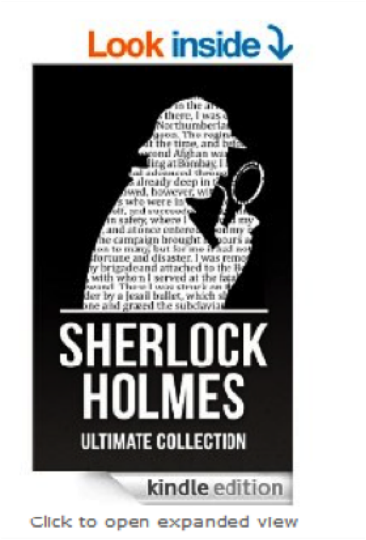
Multiclass classification

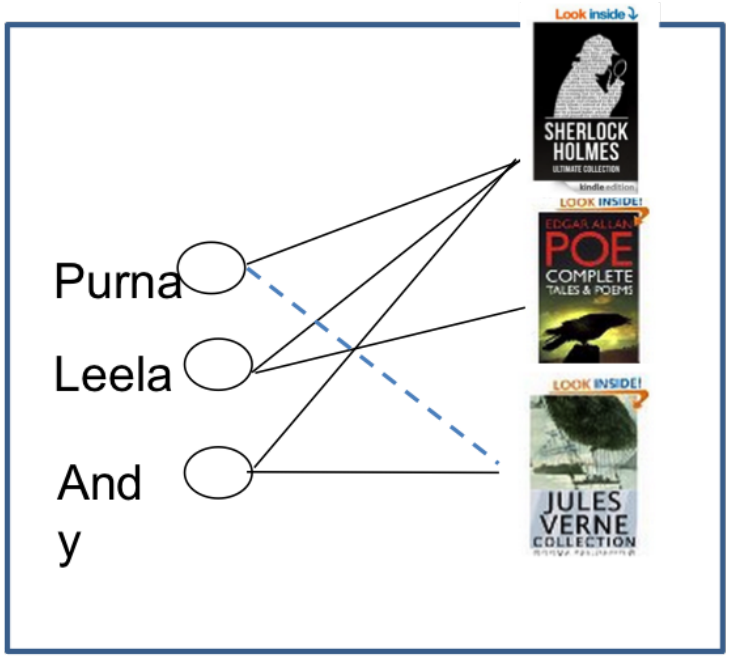
- Given n datapoints $X_1, \ldots, X_n \in \mathbb{R}^d$, and their labels $Y_1, \ldots, Y_n \in \{0, 1, \ldots, K-1\}$, output a decision function $f : \mathbb{R}^d \to \{0, 1, \ldots, K-1\}$.
- In document classification, you would have multiple categories, these categories can also have hierarchical structure.
- Goodreads has about 1K categories of books and about 2B books.

Multiclass classification

- Given n datapoints $X_1, \ldots, X_n \in \mathbb{R}^d$, and their labels $Y_1, \ldots, Y_n \in \{0, 1, \ldots, K-1\}$, output a decision function $f : \mathbb{R}^d \to \{0, 1, \ldots, K-1\}$.
- In document classification, you would have multiple categories, these categories can also have hierarchical structure.
- Goodreads has about 1K categories of books and about 2B books.
- Traditionally one would train K one-vs-all binary classifiers. If one classifier took $30s$, training $1K$ classifiers would take about 8 hours.

How about unsupervised learning– Recommender systems





Will Purna read Jules Verne?

Recommender systems

Random walk based methods

- Goal: to return top K movie recommendations for an user
- Think of netflix as a bipartite graph of users and movies.
- You may want to compute personalized pagerank from this user to all movies.

Random walk based methods

- Goal: to return top K movie recommendations for an user
- Think of netflix as a bipartite graph of users and movies.
- You may want to compute personalized pagerank from this user to all movies.
- Pagerank computation is basically computing a second eigenvector of a matrix.
- Computing it naively may take $O(nnz)$ time optimistically, where nnz is the number of non-zero entries in the matrix.

Random walk based methods

- Goal: to return top K movie recommendations for an user
- Think of netflix as a bipartite graph of users and movies.
- You may want to compute personalized pagerank from this user to all movies.
- Pagerank computation is basically computing a second eigenvector of a matrix.
- Computing it naively may take $O(nnz)$ time optimistically, where nnz is the number of non-zero entries in the matrix.
- Netflix has about 1B users and about 10K shows.
- If an user has watched about 1000 shows, then there are roughly 1000B non-zero entries in this network.

Random walk based methods- challenges

- Computing an eigen-decomposition of such a large matrix is time consuming.
- In fact if one FLOP takes 10^{-12} seconds, then one power iteration would take about 1 second.

Random walk based methods- challenges

- Computing an eigen-decomposition of such a large matrix is time consuming.
- In fact if one FLOP takes 10^{-12} seconds, then one power iteration would take about 1 second.
- If we do about 10 iterations to get an approximate solution, thats $10s$ right there.
- If you compute recommendations for one-percent of users, then 1200 years.

Random walk based methods- challenges

- Computing an eigen-decomposition of such a large matrix is time consuming.
- In fact if one FLOP takes 10^{-12} seconds, then one power iteration would take about 1 second.
- If we do about 10 iterations to get an approximate solution, thats $10s$ right there.
- If you compute recommendations for one-percent of users, then 1200 years.
- Not a good idea, since the network is also changing often, so if there are new ratings, you will need to recompute everything again.
- What can we do: caching tricks, local algorithms, randomized algorithms for matrix vector multiplications.