

# Data Mining through Spectral Analysis

Frank McSherry  
Microsoft Research, SVC  
`mcsberry@microsoft.com`

# A Matrix Trick

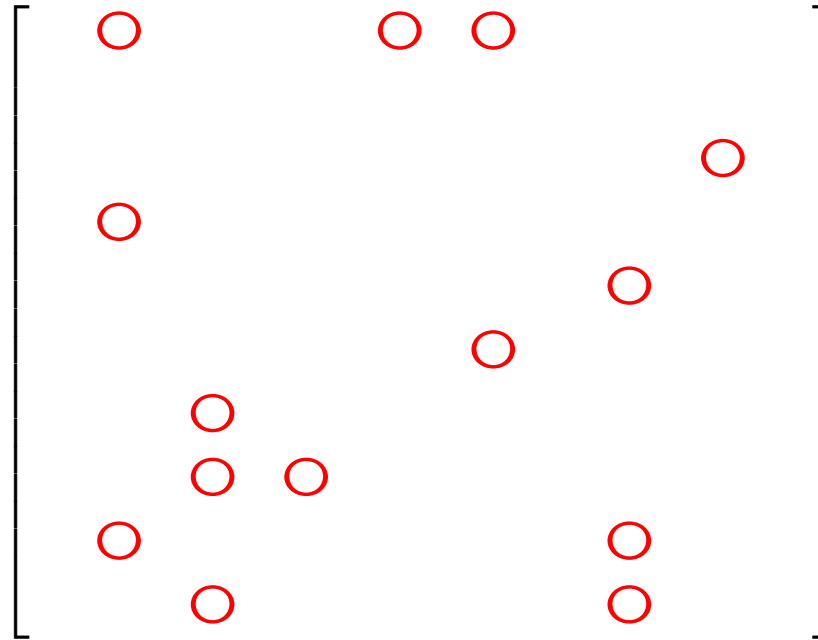
Let's imagine that you secretly choose two matrices that are tall and skinny, and multiply them:

$$\begin{bmatrix} \phantom{0} \\ \phantom{0} \\ \phantom{0} \\ \phantom{0} \end{bmatrix} \times \begin{bmatrix} \phantom{0} & \phantom{0} & \phantom{0} & \phantom{0} \end{bmatrix} = \begin{bmatrix} \phantom{0} & \phantom{0} & \phantom{0} & \phantom{0} \\ \phantom{0} & \phantom{0} & \phantom{0} & \phantom{0} \\ \phantom{0} & \phantom{0} & \phantom{0} & \phantom{0} \\ \phantom{0} & \phantom{0} & \phantom{0} & \phantom{0} \end{bmatrix}$$

You put this big matrix in your pocket, where no one can see it.

# A Matrix Trick

You now let me look at 5% of the entries, chosen randomly.



## The Trick:

By looking at 5% of the entries, I can predict 99% of the remaining entries with less than 1% error.

# Talk Outline

1. A Matrix Trick
2. Collaborative Filtering
3. A Framework for Data Mining
4. More Applications
  - Graph Partitioning
  - Web Search
  - Fast Eigencomputation
5. Conclusions / Future Directions

# Collaborative Filtering

Consider a collection of movie ratings:

Each pair (*person, movie*) will yield a rating. (ground truth)

All ratings “exist”, but we can only observe some of the entries.

**True Data** ( $D$ )

$$\begin{bmatrix} 3 & 2 & 1 & 3 \\ 2 & 4 & 4 & 2 \\ 1 & 2 & 3 & 1 \\ 4 & 2 & 1 & 3 \end{bmatrix}$$

→

**Observed Data** ( $D^?$ )

$$\begin{bmatrix} 3 & 2 & ? & 3 \\ ? & 4 & 4 & 2 \\ 1 & 2 & ? & ? \\ ? & 2 & ? & 3 \end{bmatrix}$$

**Goal:** Produce the unknown values of all ? entries.

# Collaborative Filtering

Collaborative Filtering has many applications:

- Recommendation: Determine which movies to see
- E-Commerce: Predicting prices/utility for products

## Previous Research:

Much Empirical Work: [Basu, Hirsh, Cohen], [Shardanand, Maes], [Bill-sus, Pazzani], [Breese, Heckerman, Kadie], [Pennock, Horvitz, Giles], ...

Theoretical Work: [Kumar, Raghavan, Rajagopalan, Tomkins], [Drineas, Kerenidis, Raghavan], [Kleinberg, Sandler]

# Collaborative Filtering

When can we hope to gain useful information about  $D$ ?

Without assumptions we can not:

$$\begin{bmatrix} 3 & 2 & ? & 3 \\ ? & 4 & 4 & 2 \\ 1 & 2 & ? & ? \\ ? & 2 & ? & 3 \end{bmatrix}$$

So, we will make some assumptions: [we will relax them later]

- The data matrix  $D$  has **low rank**. (latent linear structure)
- A probabilistic omission process. (not adversarial)

# Low Rank Assumption

Imagine a set of  $k$  attributes characterizing movie preferences:

$\{Humor, Violence, Romance\}$

**People:** Amos  $[0,9,0]$ , Anna  $[1,0,1]$ , Frank  $[0,0,2]$ , ...

**Movies:** Aliens  $[1,2,0]$ , Amelie  $[1,0,2]$ , Star Wars  $[2,2,2]$ , ...

The matrix  $D$  is **rank  $k$**  iff there is a  $k$ -tuple for each person and movie so that entries in  $D$  are the corresponding inner products:

$$\begin{aligned} D[Anna, Aliens] &= \langle [1, 0, 1], [1, 2, 0] \rangle \\ &= 1 + 0 + 0 \end{aligned}$$

**Important:** We only assume that  $k$  attributes exist; we do not need to know them.

# Empirical Justification

Many successful algorithms use only low rank structure:

- Latent Semantic Analysis: [Deerwester, *et al*]
- Google's PageRank: [Brin, Page]
- Spectral Clustering: [Fiedler], ...

Each approach reduces a data set to a low rank matrix and produces excellent results.

**Conclusion:** Low rank structure is meaningful.

# Random Omission Assumption

To simplify analysis:

We assume each  $D_{ij}$  is present independently with probability  $p$ .

$$D_{ij}^? = \begin{cases} D_{ij} & \text{with probability } p \\ ? & \text{otherwise} \end{cases}$$

**Important:** We have no control over which entries are available.

# An Algorithm

We now look at an algorithm for recovering data.

Algorithm **CF**( $D^?$ ):

1. Given the probability  $p$ , define

$$\widehat{D}_{ij} = \begin{cases} D_{ij}^?/p & \text{if } D_{ij}^? \neq ? \\ 0 & \text{if } D_{ij}^? = ? \end{cases}$$

2. Return  $\widehat{D}^{(k)}$ , the optimal rank  $k$  approximation to  $\widehat{D}$ .

**Note:** Computing an optimal rank  $k$  approximation can be done efficiently. We will see a fast way to do this later in the talk.

# CF Performance

How well does **CF** perform?

**Theorem 1** *Assume that the preference matrix  $D$  has rank  $k$ . Furthermore, assume that  $p$  and  $k$  are fixed. **CF** reconstructs a  $1 - o(1)$  fraction of data with  $o(1)$  error.*

One of the first collaborative filtering algorithms with strong provable performance. [Azar, Fiat, Karlin, M, Saia], *STOC 01*

**Extensions:** The algorithm (and theorem) generalize to

- $p$  and  $k$  are not fixed.
- $D$  is not rank  $k$ , but rather only has a spectral gap.
- Probability of omission is not uniform ( $p_{ij}$  for each  $i, j$ ).

# A Framework for Data Mining

The three reasons that **CF** works so well are

1. There exists a rank  $k$  matrix,  $A$ .  
The preference matrix  $D$  is rank  $k$ .
2. We wish to apply a “simple” algorithm to  $A$ .  
We just want to look at the entries of  $D$ .
3. We can construct a matrix  $\hat{A}$  that equals  $A$  in expectation.  
We produce  $\hat{D}$  by dividing all entries by  $p$ .

We are going to show that  $A$  and  $\hat{A}^{(k)}$  are almost identical.

If we apply our algorithm to  $\hat{A}^{(k)}$ , we will get an approximately correct answer.

# Defining Some Measures

We will need to define some measurements for matrices

- The L2 Norm (or Spectral Norm):

$$\|A\|_2 = \max_{|x|=1} |Ax|$$

- The Frobenius Norm:

$$\|A\|_F^2 = \sum_{i,j} (A_{ij})^2$$

We will apply these norms to  $A - \hat{A}^{(k)}$  to measure their difference.

The Frobenius norm  $\|A - \hat{A}^{(k)}\|_F^2$  measures total squared error.

## Formalizing the Approximation

**Theorem 2** Let  $\hat{A}$  be a  $m \times n$  matrix of independent random variables, whose variances are bounded by  $\sigma^2$ . If  $A = \mathbf{E}[\hat{A}]$  is rank  $k$ , then with high probability

$$\|A - \hat{A}^{(k)}\|_F^2 \text{ is } O(k\sigma^2(m+n))$$

What does this mean for collaborative filtering?

$D$  and  $\hat{D}$  correspond to  $A$  and  $\hat{A}$ , respectively.

As there are  $mn$  entries in  $D$ , the mean squared error is at most

$$\text{avg}_{i,j} \left( D_{ij} - \hat{D}_{ij}^{(k)} \right)^2 \text{ is } O\left(k\sigma^2 \left( \frac{m+n}{mn} \right)\right)$$

# Proof Outline

We need to bound the total squared error as

$$\|A - \hat{A}^{(k)}\|_F^2 \text{ is } O(k\sigma^2(m+n))$$

**The Steps:**

1. Measure L2 distance between  $A$  and  $\hat{A}$  when  $\mathbf{E}[\hat{A}] = A$ .

$$\|A - \hat{A}\|_2 \text{ is } O(\sigma\sqrt{m+n})$$

2. Relate total squared error to L2 distance

$$\|A - \hat{A}^{(k)}\|_F^2 \text{ is } O(k\|A - \hat{A}\|_2^2)$$

## Step 1: Random Matrices

**Theorem 3 (Furedi & Komlos)** *Let  $\hat{A}$  be an  $m \times n$  matrix of independent random variables with variances bounded by  $\sigma^2$ . With [alarming] high probability*

$$\|\mathbf{E}[\hat{A}] - \hat{A}\|_2 \leq 4\sigma\sqrt{m+n}$$

In our case, as  $\mathbf{E}[\hat{A}] = A$ , we conclude that

$$\|A - \hat{A}\|_2 \leq 4\sigma\sqrt{m+n}$$

## Step 2: Spectral Stability

**Lemma 4** *Let  $A$  be a  $m \times n$  rank  $k$  matrix. For any matrix  $\hat{A}$*

$$\|A - \hat{A}^{(k)}\|_F^2 \leq 8k \|A - \hat{A}\|_2^2$$

*Proof:* From linear algebra we can bound

$$\|A - \hat{A}^{(k)}\|_F^2 \leq \text{rank}(A - \hat{A}^{(k)}) \cdot \|A - \hat{A}^{(k)}\|_2^2$$

- The rank of  $A - \hat{A}^{(k)}$  is at most  $2k$ .
- We can bound

$$\begin{aligned} \|A - \hat{A}^{(k)}\|_2 &\leq \|A - \hat{A}\|_2 + \|\hat{A} - \hat{A}^{(k)}\|_2 \\ &\leq \|A - \hat{A}\|_2 + \|\hat{A} - A\|_2 \end{aligned}$$

# A Framework for Data Mining

If it is the case that:

1. There exists a rank  $k$  matrix,  $A$ .
2. We wish to apply a “simple” algorithm to  $A$ .
3. We can construct a matrix  $\hat{A}$  that equals  $A$  in expectation.

Then we can apply our algorithm to  $\hat{A}^{(k)}$ , and bound the influence of the error using the bound:

$$\|A - \hat{A}^{(k)}\|_F^2 \text{ is } O(k\sigma^2(m+n))$$

# The Hard Work

This framework is not a silver bullet; for each problem there is work to be done.

1. There exists a rank  $k$  matrix,  $A$ .

Recognize the presence of linear structure.

2. We wish to apply a “simple” algorithm to  $A$ .

Solve the problem using the linear structure.

3. We can construct a matrix  $\hat{A}$  that equals  $A$  in expectation.

Demonstrate that one can compute  $\hat{A}$ .

Finally, work through the proof to determine the precise result.

# Talk Outline

1. A Matrix Trick
2. Collaborative Filtering
3. A Framework for Data Mining
4. More Applications
  - Graph Partitioning
  - Web Search
  - Fast Eigencomputation
5. Conclusions / Future Directions

# Graph Partitioning

Consider the problem of finding cliques, colorings, and bisections that are hidden in an otherwise random graph.

**Random Graphs:** In a standard model of random graphs, each edge is included independently, with probability  $G_{ij}$ .

If  $\hat{G}$  is the adjacency matrix ( $\hat{G}_{ij} = 1$  when  $(i, j)$  is present) then

$$\hat{G}_{ij} = \begin{cases} 1 & \text{with probability } G_{ij} \\ 0 & \text{otherwise} \end{cases}$$

A common model is to set all  $G_{ij} = p$ , for some value  $p$ . In this model, all edges are equally likely.

## Relation to Graph Theory

To ensure that a clique, coloring, or bisection exists in  $\hat{G}$ , the probabilities  $G_{ij}$  must be carefully defined.

Clique	Coloring	Bisection
$\begin{bmatrix} 1 & p \\ p & p \end{bmatrix}$	$\begin{bmatrix} 0 & p & p \\ p & 0 & p \\ p & p & 0 \end{bmatrix}$	$\begin{bmatrix} q & p \\ p & q \end{bmatrix}$

The goal of each problem is to recover the planted solution.

Each problem has been solved by spectral techniques, but each solution has been highly specialized ([Alon, Kahale 94], [Boppana 87], [Condon, Karp 99], [Alon, Krivelevich, and Sudakov 98]).

# Planted Partition Model

A partition  $\psi(\cdot)$  is hidden in the random graph:

- Each node  $i$  belongs to a part  $\psi(i)$ .
- The probability  $G_{ij}$  depends only on  $\psi(i)$  and  $\psi(j)$ .

**Example:**

$$G = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix}$$

**Problem:** Given a graph instance  $\hat{G}$ , we must recover  $\psi$ .

# Graph Partitioning

We now fit Graph Partitioning to our data mining framework:

1. There exists a rank  $k$  matrix,  $A$ .

The matrix of probabilities  $G$  is rank  $k$ .

2. We wish to apply a “simple” algorithm to  $A$ .

There is a simple algorithm to get  $\psi$  from  $G$ :

We just collect all rows that are identical into a part.

3. We can construct a matrix  $\hat{A}$  that equals  $A$  in expectation.

The matrix  $\hat{G}$  equals  $G$  in expectation.

**Theorem 5** *If the vector of probabilities for each part are sufficiently different, then given an instance  $\hat{G}$ , we recover  $\psi$  with high probability.*

# Graph Partitioning Algorithm

The algorithm for partitioning based on  $G$  is easy:

Algorithm **Partition**( $G$ ):

1. While there is an unpartitioned node  $u$ :
  - (a) Form a new part including all  $\{v : G_v = G_u\}$

The algorithm for partitioning  $\hat{G}$  is nearly identical:

Algorithm **Partition**( $\hat{G}, k, \tau$ ):

1. While there is an unpartitioned node  $u$ :
  - (a) Form a new part including all  $\{v : |\hat{G}_v^{(k)} - \hat{G}_u^{(k)}| \leq \tau\}$

# Graph Partitioning [M 01]

Single simple algorithm unifies several hard problems:

- Planted Clique
- Planted  $k$ -Coloring
- Planted Bisection

Results are competitive with previous specialized approaches.

Graph Partitioning algorithm gives new results for

- Weighted Graphs (non 0/1 matrices)
- Directed Graphs (non-symmetric matrices)
- Bipartite Graphs (non-square matrices)

# Web Search

There are many web search / ranking algorithms (several of which are spectral: PageRank [Brin, Page], HITS [Kleinberg]) but we do not have a theoretical framework for comparison.

**One Approach:** Model the web and query process:

- **Link Generation** - When/why does one page link to another
- **Text Content** - Which terms appear on a page
- **Query Content** - Which terms are used in a query

Given such a model, we can define a “right answer” and compare algorithms analytically.

# Web Search Contributions

Results are from [Achlioptas, Fiat, Karlin, M, 01]

We give a natural model based on latent linear structure that helps to explain the success of HITS and Google.

Also gives a new web search algorithm that

- Has provable performance guarantees
- Unifies link/term analysis
- Generalizes HITS and Google (each use  $k = 1$ )

Interested in working on an implementation for empirical results.

# Fast Eigencomputation

In each application thus far (and in *many* other applications) we need to compute a low rank matrix approximation.

Computing  $A^{(k)}$  is similar to computing eigenvectors and singular vectors. Much work has been done on doing this quickly.

- Full Eigendecomposition  $O(mn^2)$
- Orthogonal Iteration  $O(kz \log(mn))$
- Lanczos Iteration  $O(kz \log(mn))$

where  $z$  is the number of non-zero entries.

# Near-Optimal Approximations

Recent work on near-optimal Frobenius approximations.

$$\|A - B^{(k)}\|_F^2 \leq \|A - A^{(k)}\|_F^2 + \epsilon \|A\|_F^2$$

- Frieze, Kannan, and Vempala  $O(k^{12}/\epsilon^9)$
- Drineas, Frieze, Kannan, Vempala, and Vinay  $O(nk^2/\epsilon^4)$

---

[Achlioptas, M, 01]: By properly choosing a random matrix  $\hat{A}$ ,

**Theorem 6** *We can compute  $\hat{A}^{(k)}$  in time  $O(pkz \log n)$ , where*

$$\|A - \hat{A}^{(k)}\|_2 \leq \|A - A^{(k)}\|_2 + O\left((n/p)^{1/2}\right)$$
$$\|A - \hat{A}^{(k)}\|_F \leq \|A - A^{(k)}\|_F + O\left((n/p)^{1/4} \|\hat{A}^{(k)}\|_F^{1/2}\right)$$

# Random Sparsification

Algorithm **Sparsify**( $A$ ):

1. Let  $\hat{A}$  be defined entrywise as

$$\hat{A}_{ij} = \begin{cases} A_{ij}/p & \text{with probability } p \\ 0 & \text{otherwise} \end{cases}$$

2. Compute and return  $\hat{A}^{(k)}$ .

- $\hat{A}$  equals  $A$  in expectation, and has bounded variance.
- $\hat{A}$  has  $p$  times as few non-zero entries as  $A$ .

If we apply Orthogonal Iteration to  $\hat{A}$  it will go  $1/p$  times as fast.

# Random Quantization

Algorithm **Quantize**( $A$ ):

1. Assume that  $|A_{ij}| \leq 1$ . Let  $\hat{A}$  be defined entrywise as

$$\hat{A}_{ij} = \begin{cases} 1 & \text{with probability } 1/2 + A_{ij}/2 \\ -1 & \text{with probability } 1/2 - A_{ij}/2 \end{cases}$$

2. Compute and return  $\hat{A}^{(k)}$ .

- $\hat{A}$  equals  $A$  in expectation, and has bounded variance.
- Entries in  $\hat{A}$  require only one bit to store.

We compress the space required for the matrix substantially.

# Sparsification Application

We can apply these acceleration techniques to any problem that fits the data mining framework.

Each result from the framework implicitly takes the form:

There is a threshold  $T$  such that if the variances of the entries in  $\hat{A}$  are at most  $T$ , we can substitute  $\hat{A}^{(k)}$  for  $A$ .

If we sparsify  $\hat{A}$ , the variances increase by a factor of  $1/p$ .

We can decrease  $p$  until the variances reach the limit  $T$ .

# Eigencomputation Conclusions

First algorithm for near-optimal eigencomputation that has strong spectral norm bounds. ([DKFVV] now have L2 bounds)

Algorithm runs in sub-linear time (it needn't examine the entire data set)

We can apply this acceleration trivially to each of

- Collaborative Filtering
- Web Searching
- Graph Partitioning

Applies equally well to numerous other eigenproblems.

# Data Mining Conclusions

We have introduced a framework for data mining based on latent linear structure.

We have seen how this framework leads to several rigorous applications of spectral analysis:

- Collaborative Filtering
- Graph Partitioning
- Web Searching
- Fast Eigencomputation
- Kernel PCA [Achlioptas, M, Schoelkopf], *NIPS 01*

# Practical Work

Empirical testing is the most important work to do:

- **Collaborative Filtering**

A toy system for UW area lunch venues **existed**

`http://www.cs.washington.edu/homes/mcsherry/lunch.html`

but no testing on a large scale yet.

- **Web Search** (in progress, with Anna Karlin)

- **Graph Partitioning**

Theoretical work only, but Matlab code exists!

- **Fast Low Rank Approximation**

Matlab code exists. Tested on data sets of small images where 95% omission was benign.

Recently compared with [DFKVV] on kernel and text data.

# Recent Work: Normalization

Neat case of a Practice  $\rightarrow$  Theory  $\rightarrow$  Practice cycle.

1. The degrees in most real world graphs are not uniform.
2. For Bernoullis, the variance is pretty much the expectation.  
High degree nodes yield high variance entries (Mihail & Papa)
3. Weighting links by the root of the associated nodes' degrees counteracts this. Variances become uniform again.
4. Gives good results for problems we looked at.  
Seems like a good idea to apply it to all spectral methods.

## Present Work: “Tensors”

Matrices can be thought of as functions of two variables:

$$\text{Matrix}[\text{index}_1, \text{index}_2] = \text{data}$$

Tensors are functions of any number parameters

$$\text{Tensor}[\text{index}_1, \text{index}_2, \dots, \text{index}_j] = \text{data}$$

Eg: web link data could be recorded as triples (*source, dest, text*).

---

There are models for “low rank” tensors, and some first attempts at recovering low rank approximations from corrupted data.

With the above example, we could answer queries like:

Where would page  $p$  most likely link using term  $t$ ?

## Ongoing Work: Kernels

One exciting area is Kernel PCA [Schoelkopf, Smola, Muller]:

By applying non-linear functions to the input data  $\hat{A}$  we can “straighten” non-linear trends into linear trends that we can find within our framework.

Straightening occurs by replacing the “inner product” function by another function. (the “kernel”)

Kernel PCA enables many neat new spectral applications:

- Non-linear analysis
- Analysis of non-numeric data (text, images, etc...)