The CUR Matrix Decomposition and its Applications to Algorithm Design & Massive Data Sets

Michael W. Mahoney

(joint work with Petros Drineas and Ravi Kannan)

Department of Mathematics
Yale University

michael.mahoney@yale.edu
http://cs-www.cs.yale.edu/homes/mmahoney/

Motivation

In many applications large matrices appear (too large to store in RAM).

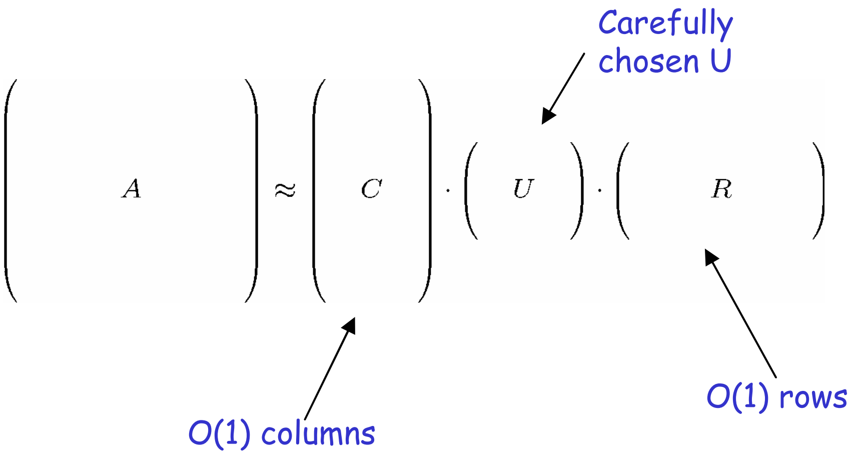
- We can make a few "passes" (sequential READS) through the matrices.
- We can create and store a small "sketch" of the matrices in RAM.
- Computing the "sketch" should be a very fast process.

Discard the original matrix and work with the "sketch".

Our approach & our results

1. A “sketch” consisting of a few rows/columns of the matrix is adequate for efficient approximations.
2. We draw the rows/columns randomly, using importance sampling. Rows/columns are picked with probability proportional to their lengths.

Create an approximation to the original matrix which can be stored in much less space.



$$A \approx C \cdot U \cdot R$$

Carefully chosen U

O(1) columns

O(1) rows

Overview

- Motivation
- The CUR decomposition
  - Our decomposition, error bounds
  - Alternative constructions for U (sketch)
- Applications
  - Approximating Weighted Max-Cut and a Linear Programming result
  - Fast computation of kernel decompositions for Machine Learning
  - Recommendation Systems
- A TensorCUR extension
  - Recommendation Systems Revisited: a tensor model
- Conclusions

Applications: Data Mining

We are given m ($>10^6$) objects and n($>10^5$) features describing the objects.

Database

An m-by-n matrix *A* (A_{ij} shows the "importance" of feature j for object i).

E.g., m documents, represented w.r.t. n terms.

Queries

Given a new object x, find similar objects in the database (*nearest neighbors*).

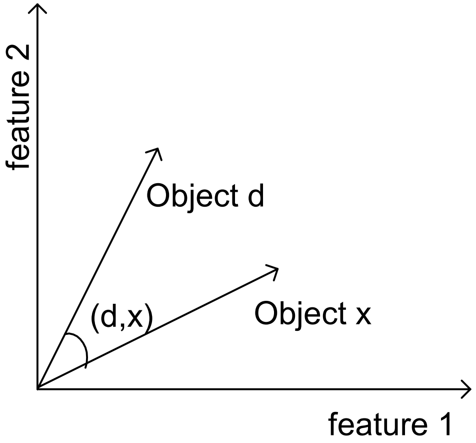
Applications (cont'd)

Two objects are "close" if the angle between their corresponding (normalized) vectors is small. So,

$$x^T \cdot d = \cos(x,d)$$

is high when the two objects are close.

$A \cdot x$ computes all the angles and answers the query.



Key observation: The exact value $x^T \cdot d$ might not be necessary.

1. The feature values in the vectors are set by coarse heuristics.
2. It is in general enough to see if $x^T \cdot d >$ **Threshold**.

Using the CUR decomposition

Assume that CUR is an approximation to A, such that CUR is stored efficiently (e.g. in RAM).

Given a query vector x, instead of computing A · x, compute CUR · x to identify its nearest neighbors.

The CUR algorithm guarantees a bound on the worst case choice of x.

$$\max_{x:|x|=1} \|Ax - CURx\| = \|A - CUR\|_2 \leq \epsilon \|A\|_F$$

Overview

- Motivation
- The CUR decomposition

  Our decomposition

  Alternative constructions for U (sketch)
- Applications

  Approximating Weighted Max-Cut and a Linear Programming result

  Fast computation of kernel decompositions for Machine Learning

  Recommendation Systems
- A TensorCUR extension

  Recommendation Systems Revisited: a tensor model
- Conclusions

Our CUR decomposition

Given a **large** m-by-n matrix A (stored on disk), compute a decomposition CUR of A such that:

1. C, U, R can be stored in O(m+n) space, after making **two** passes through the entire matrix A, using O(m+n) additional space and time.

2. The product CUR satisfies (with high probability)

$$\|A - CUR\|_2 \le \epsilon \|A\|_F$$

$$\|A\|_F^2 = \sum_{i,j} A_{ij}^2 \quad , \quad \|A\|_2 = \max_{x:|x|=1} |Ax|$$

(and a similar bound with respect to the Frobenius norm).

Describing A' = C · U · R

$$\underset{m \times n}{\begin{pmatrix} \\ \\ & & A & & \\ \\ \\ \end{pmatrix}} \approx \underset{m \times c}{\begin{pmatrix} \\ \\ & C & \\ \\ \\ \end{pmatrix}} \cdot \underset{c \times r}{\begin{pmatrix} & U & \end{pmatrix}} \cdot \underset{r \times n}{\begin{pmatrix} & & R & & \end{pmatrix}}$$

C consists of c = O(1/ε^2) columns of A and R consists of r = O(1/ε^2) rows of A (the "description length" of A is O(m+n)).

C (and R) is created using importance sampling, e.g. columns (rows) are picked in i.i.d. trials with respect to probabilities

$$p_i = |A^{(i)}|^2 / \sum_i |A^{(i)}|^2$$

Computing U

Intuition:

The CUR algorithm essentially expresses every row of the matrix A as a linear combination of a small subset of the rows of A.

- This small subset consists of the rows in R.
- Given a row of A – say $A_{(i)}$ – the algorithm computes a good fit for the row $A_{(i)}$ using the rows in R as the basis, by approximately solving

$$\min_u \left\| \underset{1\times n}{\left(\qquad A_{(i)} \qquad \right)} - \underset{1\times r}{\left(\quad u \quad \right)} \cdot \underset{r\times n}{\left(\qquad R \qquad \right)} \right\|_2$$

Notice that only c = O(1) element of the i-th row are given as input.

However, a vector of coefficients *u* can still be computed.

Computing U (cont'd)

Given *c* elements of $A_{(i)}$ the algorithm computes a good fit for the row $A_{(i)}$ using the rows in R as the basis, by approximately solving:

$$\min_u \left\| \underset{1 \times c}{\left(\quad \tilde{A}_{(i)} \quad \right)} - \underset{1 \times r}{\left(\quad u \quad \right)} \cdot \underset{r \times c}{\left(\quad \tilde{R} \quad \right)} \right\|_2$$

However, our CUR decomposition *approximates* the vectors *u* instead of exactly computing them.

Open problem: Is it possible to improve our error bounds using the optimal coefficients?

Singular Value Decomposition (SVD)

$$\underset{m \times n}{\begin{pmatrix} & & \\ & A & \\ & & \end{pmatrix}} = \underset{m \times m}{\begin{pmatrix} & & \\ & U & \\ & & \end{pmatrix}} \cdot \underset{m \times n}{\begin{pmatrix} & & \\ & \Sigma & \\ & & \end{pmatrix}} \cdot \underset{n \times n}{\begin{pmatrix} & & \\ & V & \\ & & \end{pmatrix}}^{T}$$

U (V): orthogonal matrix containing the left (right) singular vectors of A.

Σ: diagonal matrix containing the singular values of A.

1. Exact computation of the SVD takes O(min{mn^2 , m^2n}) time.
2. The top few singular vectors/values can be approximated faster (Lanczos/ Arnoldi methods).

Rank k approximations (A_k)

$$\underset{m \times n}{\left(\quad A_k \quad \right)} = \underset{m \times k}{\left(\quad U_k \quad \right)} \cdot \underset{k \times k}{\left(\quad \Sigma_k \quad \right)} \cdot \underset{k \times n}{\left(\quad V_k^T \quad \right)}$$

U_k (V_k): orthogonal matrix containing the top k left (right) singular vectors of A.

Σ_k: diagonal matrix containing the top k singular values of A.

A_k is a matrix of rank k such that $||A-A_k||_{2,F}$ is minimized over all rank k matrices!

This property of very useful in the context of Principal Component Analysis.

Fixing the rank of U

In the process of computing *U* , we **fix its rank** to be a positive constant *k*, which is part of the input.

Note: Since CUR is of rank k, $||A\text{-}CUR||_{2,F} > ||A\text{-}A_k||_{2,F}$.

Thus, we should choose a k such that $||A\text{-}A_k||_{2,F}$ is small.

Error bounds (Frobenius norm)

Assume A_k is the “best” rank k approximation to A (through SVD). Then

$$\mathbf{E}\left(\|A - CUR\|_F^2\right) \;\leq\; \|A - A_k\|_F^2 + \varepsilon\|A\|_F^2$$

We need to pick $O(k/\varepsilon^2)$ rows and $O(k/\varepsilon^2)$ columns.

Error bounds (2-norm)

Assume A_k is the "best" rank k approximation to A (through SVD). Then

$$\begin{aligned}
\mathbf{E}\left(\|A - CUR\|_2^2\right) &\leq \|A - A_k\|_2^2 + \varepsilon\|A\|_F^2 \\
&\leq \left(\frac{1}{k+1} + \varepsilon\right)\|A\|_F^2 \\
&\leq 2\varepsilon\|A\|_F^2
\end{aligned}$$

We need to pick $O(1/\varepsilon^2)$ rows and $O(1/\varepsilon^2)$ columns and set $k = (1/\varepsilon)-1$.

Other CUR decompositions (1)

Computing U in constant time (instead of O(m+n))

Our CUR decomposition computes a provably efficient U in linear time.

In recent work (DM '04), we demonstrate how to compute a provably efficient U in constant time – the ConstantTimeCUR decomposition.

Our ConstantTimeCUR decomposition:

- samples $O(\text{poly}(k,\varepsilon))$ rows and columns of A,
- needs an extra pass through the matrix A,
- significantly improves the error bounds of Frieze, Kannan, and Vempala, FOCS '98, JACM '04,
- is useful for designing approximation algorithms,
- but has a more complicated analysis.

Other CUR decompositions (2)

Solving for the optimal U

Given *c* elements of the i-th row of A, the algorithm computes the "best fit" for the i-th row using the rows in R as the basis, by solving:

$$\min_u \left\| \underset{1 \times c}{\left(\quad \tilde{A}_{(i)} \quad \right)} - \underset{1 \times r}{\left(\quad u \quad \right)} \cdot \underset{r \times c}{\left(\quad \tilde{R} \quad \right)} \right\|_2 \longrightarrow \quad u = \tilde{R}^{+} \tilde{A}_{(i)}$$

Using the above strategy, we can also compute a CUR decomposition, with a different U in the middle.

(This decomposition has been experimentally proposed in the context of fast kernel computation.)

Open problem: What is the improvement?

Other CUR decompositions (2,cont'd)

An alternative perspective:

$$\begin{pmatrix} \quad A \quad \end{pmatrix} \approx \begin{pmatrix} C \end{pmatrix} \cdot \begin{pmatrix} U \end{pmatrix} \cdot \begin{pmatrix} \quad R \quad \end{pmatrix}$$

Optimal U

Q. Can we find the "best" set of columns and rows to include in C and R?

Randomized and/or deterministic strategies are acceptable.

Results by S. A. Goreinov, E. E. Tyrtyrshnikov, and N.L. Zamarashkin imply (rather weak) error bounds if we choose the columns and rows of A that define a parallelpiped of maximal volume.

Overview

- Motivation
- The CUR decomposition
  - Our decomposition
  - Alternative constructions for U (sketch)
- Applications
  - Approximating Weighted Max-Cut and a Linear Programming result
  - Fast computation of kernel decompositions for Machine Learning
  - Recommendation Systems
- A TensorCUR extension
  - Recommendation Systems Revisited: a tensor model
- Conclusions

Approximating Max-Cut

Max-Cut (NP-hard for general and dense graphs, Max-SNP)

Given a graph $G=(V,E)$, $|V|=n$, partition V in two disjoint subsets V_1 and V_2 such that the number of edges of E that have one endpoint in V_1 and one endpoint in V_2 is maximized.

Goemans & Williamson '94: .878-approximation algorithm **(might be tight!)**

Arora, Karger & Karpinski '95: LP based, εn^2 additive error(*)

De La Vega '96: Combinatorial methods, εn^2 additive error

Frieze & Kannan '96: Linear Algebraic methods, εn^2 additive error(*)

Goldreich, Goldwasser & Ron '96: Property Testing, εn^2 additive error

Alon, De La Vega, Kannan & Karpinski '02, '03 : Cut Decomposition, εn^2 additive error(*)

All the above algorithms run in constant time/space.

(*) More generally, it achieves εn^r for all problems in Max-r-CSP (=Max-SNP)

Our result for Weighted Max-Cut

Let A denote the adjacency matrix of G. All previous algorithms also guarantee an

$$\varepsilon n^2 A_{\text{max}}$$

additive error approximation for the **weighted** Max-Cut problem, where A_{max} is **the maximum edge weight** in the graph.

Our result:

We can approximate the **weighted** Max-Cut of G(V,E) up to additive error

$$\varepsilon n^2 A_{\text{avg}} = \epsilon n^2 \frac{\|A\|_F}{n} = \epsilon n^2 \sqrt{\frac{\sum_{i,j\in E} A_{ij}^2}{n^2}}$$

in constant time and space after reading the graph a small number of times.

Our algorithm (sketch)

Let A denote the adjacency matrix of the **weighted** graph G=(V,E). Then,

$$\text{MaxCut}(A) = \max_{x\in\{0,1\}^n} x^T A(\vec{1}_n - x)$$

Step 1: Replace A by the ConstantTimeCUR decomposition. Then,

$$\left| \max_{x\in\{0,1\}^n} x^T A(\vec{1}_n - x) - \max_{x\in\{0,1\}^n} x^T CUR(\vec{1}_n - x) \right| \leq \epsilon n \|A\|_F$$

Follows from $\left| x(\vec{1} - x) \right| \leq n$

$$\|A - CUR\|_2 \leq \epsilon \|A\|_F$$

Compute the MaxCut of CUR, thus getting an approximation to the MaxCut of A.

Why is it easier to solve the problem on CUR?

Our algorithm (cont'd)

Step 2: Recall that A is an n-by-n matrix. We seek to compute

$$\text{MaxCut}(CUR) = \text{max}_{x\in\{0,1\}^n} \left(x^T C\right) U \left(R(\vec{1}_n - x)\right)$$

$$\underset{1 \times n}{\left(\quad x^T \quad \right)} \underset{n \times c}{\left(\begin{matrix} \\ C \\ \\ \end{matrix} \right)} \cdot \underset{c \times r}{\left(\; U \; \right)} \cdot \underset{r \times n}{\left(\quad R \quad \right)} \underset{n \times 1}{\left(\begin{matrix} \\ \vec{1}_n - x \\ \\ \\ \end{matrix} \right)}$$

Let $u^T = x^T C$

Let $v = R(\vec{1}_n - x)$

Our algorithm (cont'd)

Step 2: We seek to compute

$$\text{MaxCut}(CUR) = \max_{x \in \{0,1\}^n} \quad u^T U v$$

$$\begin{aligned} |u^T| &= |x^T C| \\ &\leq O(\sqrt{n}\|A\|_F) \end{aligned}$$

$$\underbrace{\begin{pmatrix} \quad u^T \quad \end{pmatrix}}_{1 \times c} \cdot \underbrace{\begin{pmatrix} \quad \\ \quad U \quad \\ \quad \end{pmatrix}}_{c \times r} \cdot \underbrace{\begin{pmatrix} \\ v \\ \\ \end{pmatrix}}_{r \times 1}$$

Notice:

$$u \in \left[-\alpha_1\sqrt{n}\|A\|_F, \alpha_1\sqrt{n}\|A\|_F\right]^c$$

$$v \in \left[-\alpha_2\sqrt{n}\|A\|_F, \alpha_2\sqrt{n}\|A\|_F\right]^r$$

"Discretize" the cubes !

Our algorithm (cont'd)

Step 2: "Discretize" the cubes by placing a grid of width

$[O(1)]^c$ choices for u ⬅ $u \in \left[-\alpha_1\sqrt{n}\|A\|_F, \alpha_1\sqrt{n}\|A\|_F\right]^c$ ⬅ $\delta = O\left(\sqrt{n}\|A\|_F\right)$

$[O(1)]^r$ choices for v ⬅ $v \in \left[-\alpha_2\sqrt{n}\|A\|_F, \alpha_2\sqrt{n}\|A\|_F\right]^r$

Overall, we will check every possible value for the **vectors** *u* and *v* on the "discretized" cubes. Notice that there are **O(1) such vectors.**

Pick the *u* and *v* that maximize this product:

$$\max_{u,v} \left(\quad u^T \quad \right) \cdot \left(\quad U \quad \right) \cdot \left(\quad v \quad \right)$$

Our algorithm (cont'd)

Notice that for *u* and *v* to be a valid solution, there must exist

$$x \in \{0,1\}^n : \begin{array}{l} u^T = x^T C \\ v = R(\vec{1} - x) \end{array}$$

Because of **discretization**, this relaxes to

$$x \in \{0,1\}^n : \begin{array}{l} \left| u^T - x^T C \right| \le O(\delta)\vec{1}_c \longleftarrow \text{c constraints, n variables} \\ \left| v - R(\vec{1} - x) \right| \le O(\delta)\vec{1}_r \longleftarrow \text{r constraints, n variables} \end{array}$$

This is an Integer Programming feasibility question, but we can relax it to **Linear Programming feasibility**, because there are far fewer constraints than variables.

More machinery …

After relaxing the IP to the LP, we (roughly) need to check the feasibility of

$$x \in [0,1]^n : \quad \begin{array}{l} \left| u^T - x^T C \right| \le O(\delta)\vec{1}_c \\ \left| v - R(\vec{1} - x) \right| \le O(\delta)\vec{1}_r \end{array}$$

Not done yet! We need to check the feasibility in constant time.

- We devised a method to (approximately) check the feasibility of a large LP using a non-uniformly randomly chosen subset of the variables.
- A similar lemma for a uniformly chosen subprogram was implicit in AFKK '02.

A Linear Programming Lemma

Let $P^{(i)}$ denote the i th column of the r-by-n matrix P and suppose that the following Linear Program is **infeasible**:

$$Px = \sum_{i=1}^{n} P^{(i)} x_i \le b \qquad 0 \le x_i \le 1$$

Form Q, a random non-uniform subset of {1…n}, with |Q|=q. Q is formed in q i.i.d. trials, where

$$p_i = \frac{|P^{(i)}|}{\sum_{i=1}^{n} |P^{(i)}|}$$

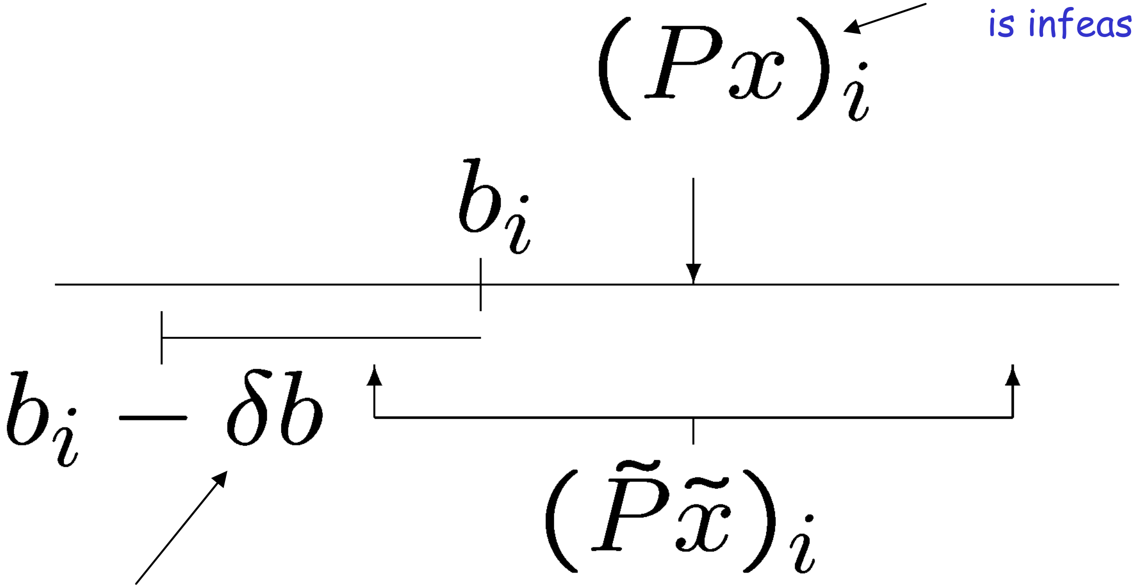
With probability at least 1-δ, the following LP is **infeasible** as well:

$$\tilde{P}\tilde{x} = \sum_{i \in Q} \frac{1}{q p_i} P^{(i)} x_i \le b - \delta b \cdot \vec{1}_r$$

$$0 \le x_i \le 1 \;,\; i \in Q$$

$$\delta b = \frac{O(\log \delta)}{\sqrt{q}} \cdot n \|P\|_F$$

The picture …



The i-th constraint is infeasible

Tightening the constraint guarantees infeasibility

Discarding variables “perturbs” the constraint

A converse LP Lemma

Let $P^{(i)}$ denote the i th column of the r-by-n matrix P and suppose that the following Linear Program is **feasible**:

$$Px = \sum_{i=1}^{n} P^{(i)} x_i \le b \qquad 0 \le x_i \le 1$$

Form Q, a random non-uniform subset of {1…n}, with |Q|=q. Q is formed in q i.i.d. trials, where

$$p_i = \frac{|P^{(i)}|}{\sum_{i=1}^{n} |P^{(i)}|}$$

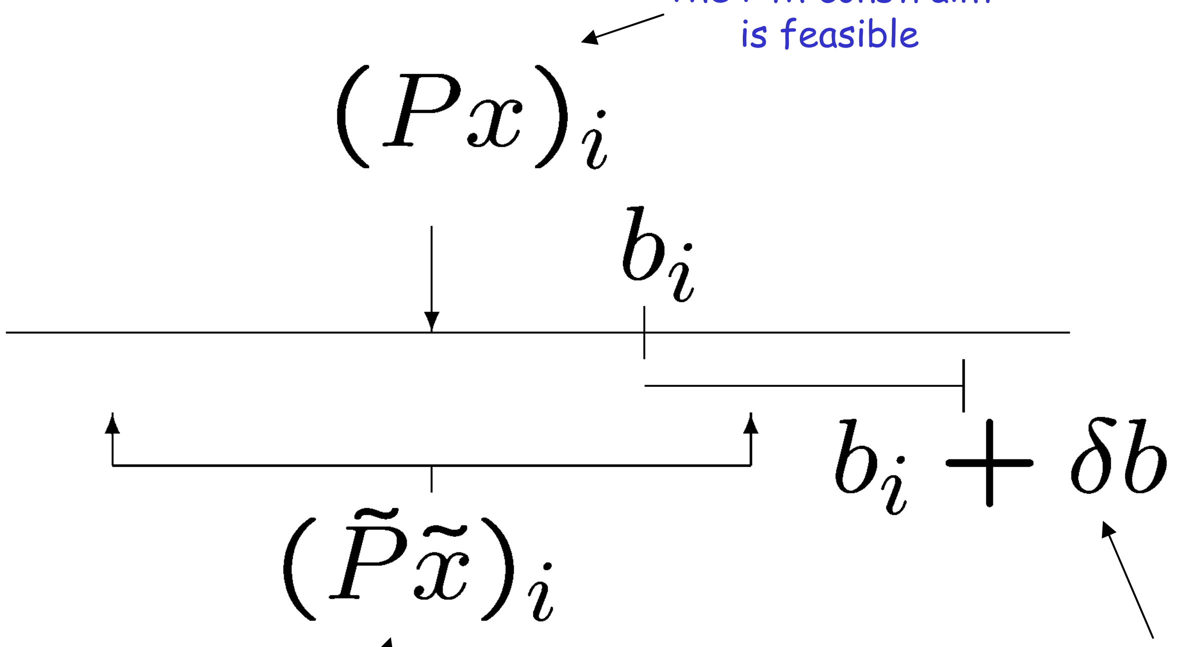
With probability at least 1-δ, the following LP is **feasible** as well:

$$\tilde{P}\tilde{x} = \sum_{i \in Q} \frac{1}{q p_i} P^{(i)} x_i \le b + \delta b \cdot \vec{1}_r$$

$$0 \le x_i \le 1 \ , \ i \in Q$$

$$\delta b = \frac{O(\log \delta)}{\sqrt{q}} \cdot n \|P\|_F$$

The picture …



The i-th constraint is feasible

Discarding variables "perturbs" the constraint

Relaxing the constraint guarantees feasibility

Fast Computation of Kernels

Q. SVD has been used to identify/extract linear structure from data. What about non-linear structures, like multi-linear structures or non-linear manifold structure?

A. Kernel-based learning algorithms.

Data $\Psi = \{\Psi_{(1)}, \ldots, \Psi_{(m)}\} \in R^{m \times n}$

Mapping $\phi : \Psi \rightarrow \Phi$ (feature space)

Gram Matrix $G_{ij} = G\left(\Psi_{(i)}, \Psi_{(j)}\right) = \left\langle \phi\left(\Psi_{(i)}\right), \phi\left(\Psi_{(j)}\right) \right\rangle$

PSD matrix inner product

Algorithms extracting linear structure can be applied to G without knowing ϕ !

Isomap, LLE, Laplacian Eigenmaps, SDE, are all Kernel PCA for special Gram matrices.

However, running, e.g., SVD to extract linear structure from the Gram matrix still requires $O(m^3)$ time.

We can apply CUR-type decompositions to speed up such calculations.

Fast Computation of Kernels (cont'd)

A potential issue is that the CUR decomposition of the Gram matrix is not a positive semidefinite matrix.

However, if we compute the "optimal" U matrix, then the CUR approximation to the optimal matrix is PSD.

For the special case of PSD matrix $G = XX^T$ for some matrix X, we can prove that using the "optimal" U guarantees:

$$\|G - CUC^T\|_F^2 \leq \|G - G_k\|_F^2 + \epsilon \|X\|_F^4$$

$\|X\|_F^4$ vs. $\|G\|_F^2 = \|XX^T\|_F^2$

Recommendation Systems

The problem:

Assume the existence of m customers and n products. Then, A is an (unknown) matrix s.t. A_{ij} is the utility of product j for customer i.

Our goal is to recreate A from a few samples, s.t. we can recommend high utility products to customers.

- R. Kumar, P. Raghavan, S. Rajagopalan & A. Tomkins '98.

(assuming strong clustering of the products, they offer competitive algorithms even with only 2 samples/customer)

- Y. Azar, A. Fiat, A. Karlin, F. McSherry & J. Saia '01.

(assuming sampling of $\Omega(mn)$ entries of A and a certain gap requirement, they -very- accurately recreate A)

Recommendation Systems

The problem:

Assume the existence of m customers and n products. Then, A is an (unknown) matrix s.t. A_{ij} is the utility of product j for customer i.

Our goal is to recreate A from a few samples, s.t. we can recommend high utility products to customers.

Critical assumption (supported by experimental evidence):

There exist a small, constant (e.g. *k*) number of different customer types and all customers are linear combinations of these types or, equivalently, the matrix A has a "good" low-rank (e.g. *k*) approximation.

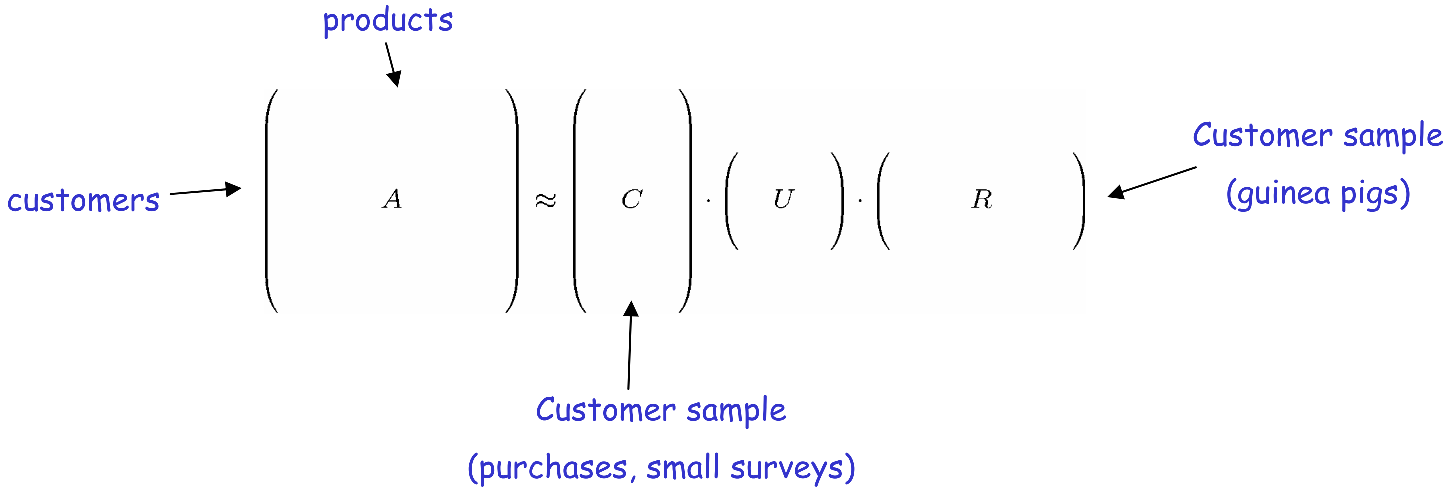
Recommendation Systems (cont'd)

Question:

Can we get competitive performance by sampling less than Ω(mn) elements?

Answer:

Apply the CUR decomposition.



products

customers

$$A \approx C \cdot U \cdot R$$

Customer sample

(guinea pigs)

Customer sample

(purchases, small surveys)

Recommendation Systems (cont'd)

Details:

Sample a constant number of rows and columns of A and compute A'=CUR. Assuming that A has a "good" low rank approximation,

$$\|A - CUR\|_F^2 \leq \|A - A_k\|_F^2 + \varepsilon\|A\|_F^2$$

This implies that we can make relatively accurate recommendations with far fewer samples.

Overview

- Motivation
- The CUR decomposition
  - Our decomposition
  - Alternative constructions for U (sketch)
- Applications
  - Approximating Weighted Max-Cut and a Linear Programming result
  - Fast computation of kernel decompositions for Machine Learning
  - Recommendation Systems
- A TensorCUR extension
  - Recommendation Systems Revisited: a tensor model
- Conclusions

Recommendation Systems Revisited

Comment:

It is folklore knowledge in economics literature that utility is an ordinal and not a cardinal quantity.

Thus, it is more natural to compare products than assign utility values.

Model revisited: m customers and n products

Every customer has an n-by-n matrix (whose entries are §1) and represent pairwise product comparisons.

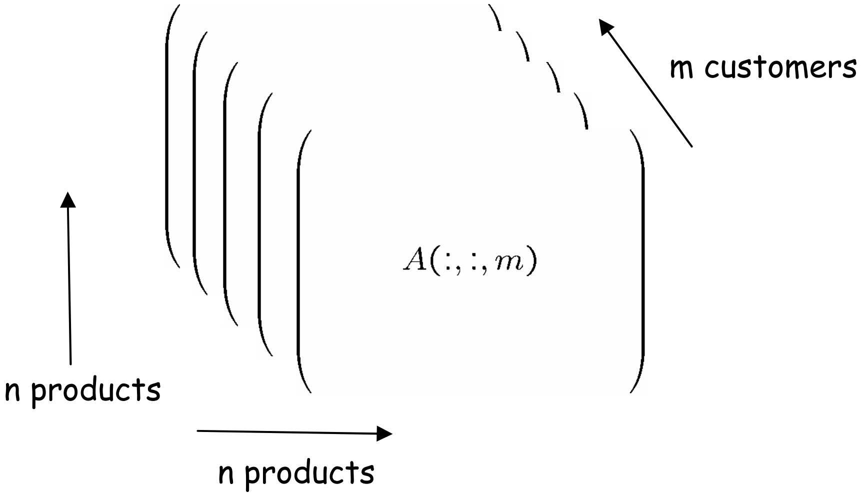
Overall, there are m such matrices, forming an n-by-n-by-m three-mode tensor (three-dimensional array), denoted by A.

We seek to extract "structure" from this tensor by generalizing the CUR decomposition.

Recommendation Systems (cont'd)

Our goal:

Recreate the tensor A from a few samples in order to recommend high utility products to the customers.



Q. What do we know about tensor decompositions?

A. Not much, although tensors arise in numerous applications.

Tensors

Tensors appear both in Math and CS.

- Represent high dimensional functions
- Connections to complexity theory (i.e., matrix multiplication complexity)
- Data Set applications (i.e., Independent Component Analysis, higher order statistics, etc.)

Also, many practical applications, e.g., Medical Imaging, Hyperspectral Imaging, video, Psychology, Chemometrics, etc.

However, there does not exist a definition of tensor rank (and associated tensor SVD) with the – nice – properties found in the matrix case.

Tensor rank

A definition of tensor rank

Given a tensor

$$A \in \mathcal{R}^{n_1 \times n_2 \times \ldots n_d}$$

find the minimum number of rank one tensors into it can be decomposed.

$$A = \sum_{i=1}^{r} u_1^i \otimes u_2^i \otimes \ldots \otimes u_d^i$$

outer product

- agrees with matrices for d=2
- related to computing bilinear forms and algebraic complexity theory.

BUT

- only weak bounds are known
- tensor rank depends on the underlying ring of scalars
- computing it is NP-hard
- successive rank one approximations are no good

Tensor a-rank

The a-rank of a tensor

Given

$$A \in \mathcal{R}^{n_1 \times n_2 \times \dots n_d}$$

create the “unfolded” matrix

$$A_{[\alpha]} \in R^{n_\alpha \times N_\alpha}$$

$$N_\alpha = \prod_{i \neq \alpha} n_i$$

and compute its rank, called the a-rank.

Pros:

- Easily computable,

Cons:

- different a-ranks are different,
- information is lost.

$$\begin{array}{cc} n_\alpha & \begin{pmatrix} \qquad\qquad & & \qquad\qquad \\ & A_{[\alpha]} & \\ & & \end{pmatrix} \\ & n_1 n_2 n_{\alpha-1} n_{\alpha+1} n_d \end{array}$$

Tensors in real applications

3 classes of tensors in data applications

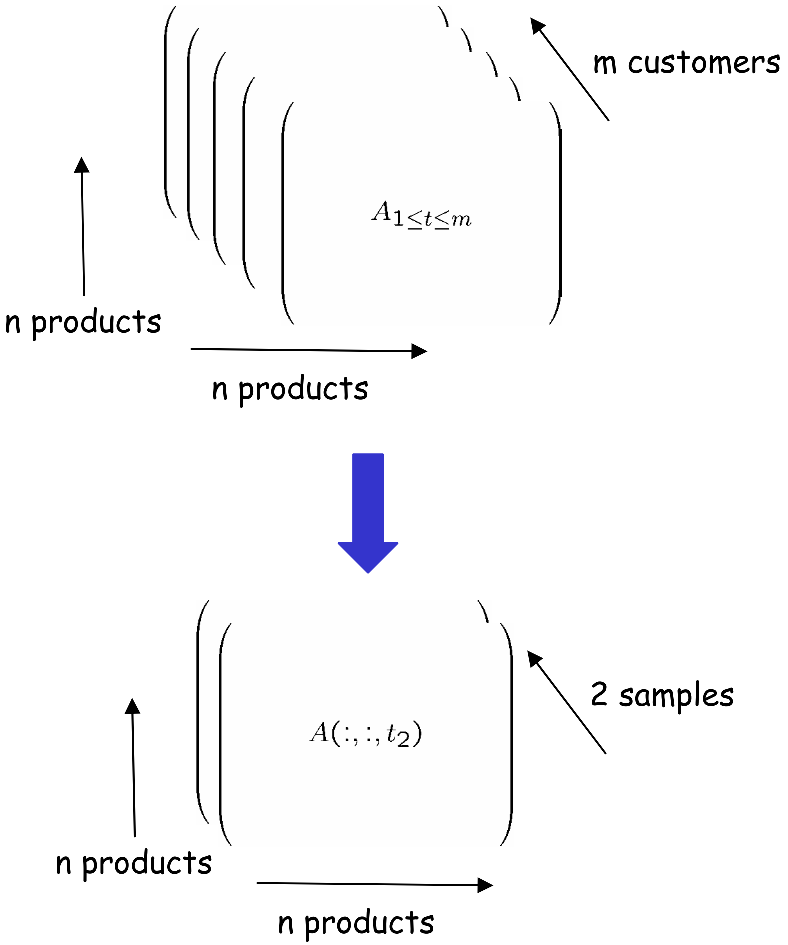
1. All modes are comparable (e.g., tensor faces, chemometrics)
2. A priori dominant mode (e.g., coarse scales, pictures vs. time)
3. All other combinations

D. and Mahoney '04, TensorSVD paper deals with (1).

D. and Mahoney '04, TensorCUR paper deals with (2).

We will focus on (2), where there is a preferred mode.

The TensorCUR algorithm (3-modes)



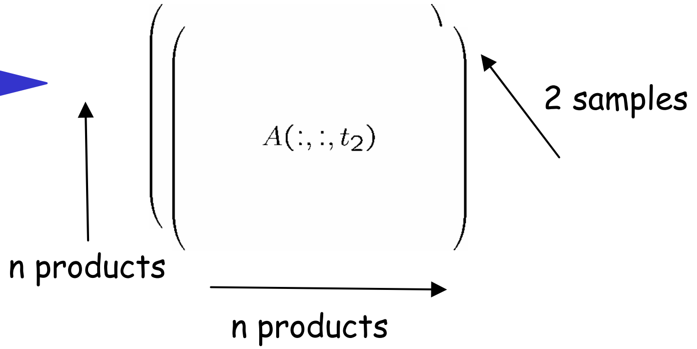
- Choose the preferred mode α (customers)
- Pick a few representative customers

$$p_t = \frac{\|A(:,:,t)\|_F^2}{\sum_{t=1}^{m} \|A(:,:,t)\|_F^2}$$

- Express all the other customers in terms of the representative customers.

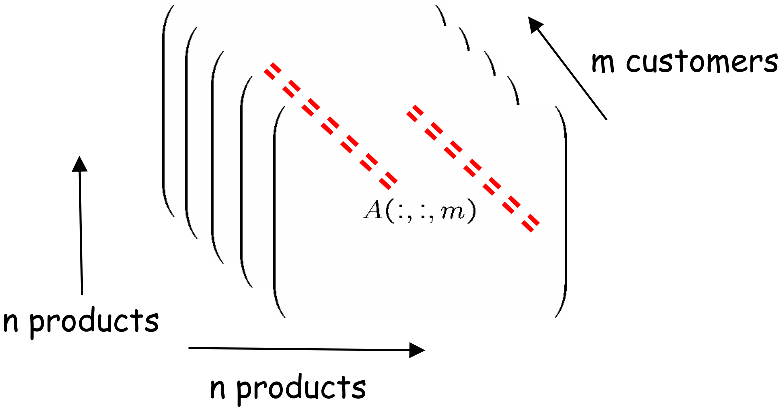
The TensorCUR algorithm (cont'd)

Let R denote the tensor of the sampled matrices



Express the matrices of the remaining customers as linear combinations of the reconstructed matrices.

- First, pick a constant number of "fibers" of the tensor A (the red dotted lines).
- Express the matrices of the remaining customers as linear combination of the (approximate) sampled matrices.



$$\min_u \sum_{i,j} \left(A(i,j,s) - \sum_{s \in R} u_s A(i,j,s)\right)^2$$

sampled fibers

sampled customers

The TensorCUR algorithm (cont'd)

Theorem:

$$\|A - CU \times_\alpha R\|_F^2 \leq \left\| A_{[\alpha]} - \left(A_{[\alpha]}\right)_{k_\alpha} \right\|_F^2 + \epsilon \|A\|_F^2$$

Unfold R along the α dimension and pre-multiply by CU

Best rank k_a approximation to $A_{[a]}$

How to proceed:

- Can recurse on each sub-tensor in R,
- or do SVD, exact or approximate,
- or do kernel-based diffusion analysis,
- or do wavelet-based methods.

Why are the subtensors of R low rank?

If customer product preferences are transitive, then the matrix is well approximated by a low-rank matrix.

Small perturbations do not affect the above property.

Conclusions

- CUR \approx A for matrices
  - Algorithmic applications: Max-CUT.
  - Machine Learning applications: Kernels.
  - Recommendation Systems.
- TensorCUR
  - Framework for dealing with very large tensor-based data sets,
  - to extract a “sketch” in a principled and optimal manner,
  - which can be coupled with more traditional methods of data analysis.