UNIVERSITY OF MINNESOTA TWIN CITIES

Efficient Linear Algebra methods for Data Mining

Yousef Saad

Department of Computer Science and Engineering

University of Minnesota

NUMAN-08 Kalamata, 2008

Team members involved in this work – Support

Past:

- Efi Kokiopoulou [Now at the U. of Lausanne]

Current:

- Jie Chen [grad student]
- Sofia Sakellaridi [grad student]
- Haw-Ren Fang [Post-Doc]

Support:

- National Science Foundation

Introduction, background, and motivation

Common goal of data mining methods: to extract meaningful information or patterns from data. Very broad area – includes: data analysis, machine learning, pattern recognition, information retrieval, ...

- Main tools used: linear algebra; graph theory; approximation theory; optimization; ...
- In this talk: emphasis on dimension reduction techniques and the interrelations between techniques

- Focus on two main problems

– Information retrieval

– Face recognition

- and 3 types of dimension reduction methods

– Standard subspace methods [SVD, Lanczos]

– Graph-based methods

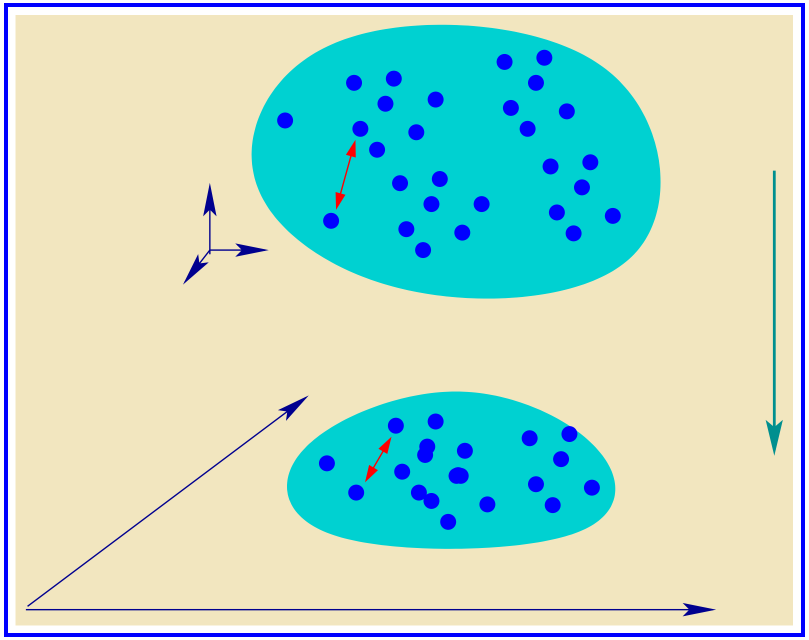
– multilevel methods

The problem

- Given $d \ll m$ find a mapping

$\Phi : x \in \mathbb{R}^m \longrightarrow y \in \mathbb{R}^d$

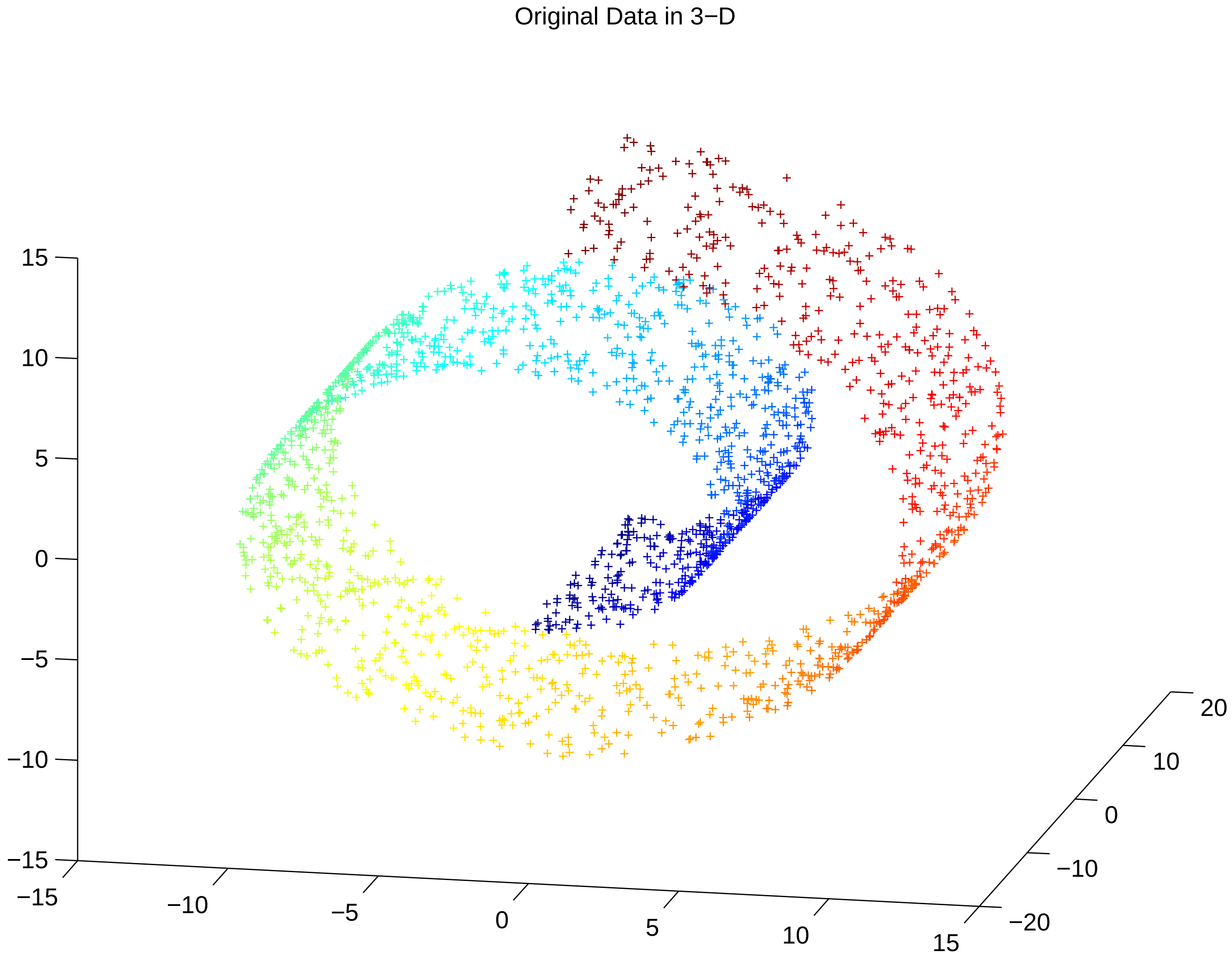
- Mapping may be explicit (e.g., $y = V^T x$)
- Or implicit (nonlinear)



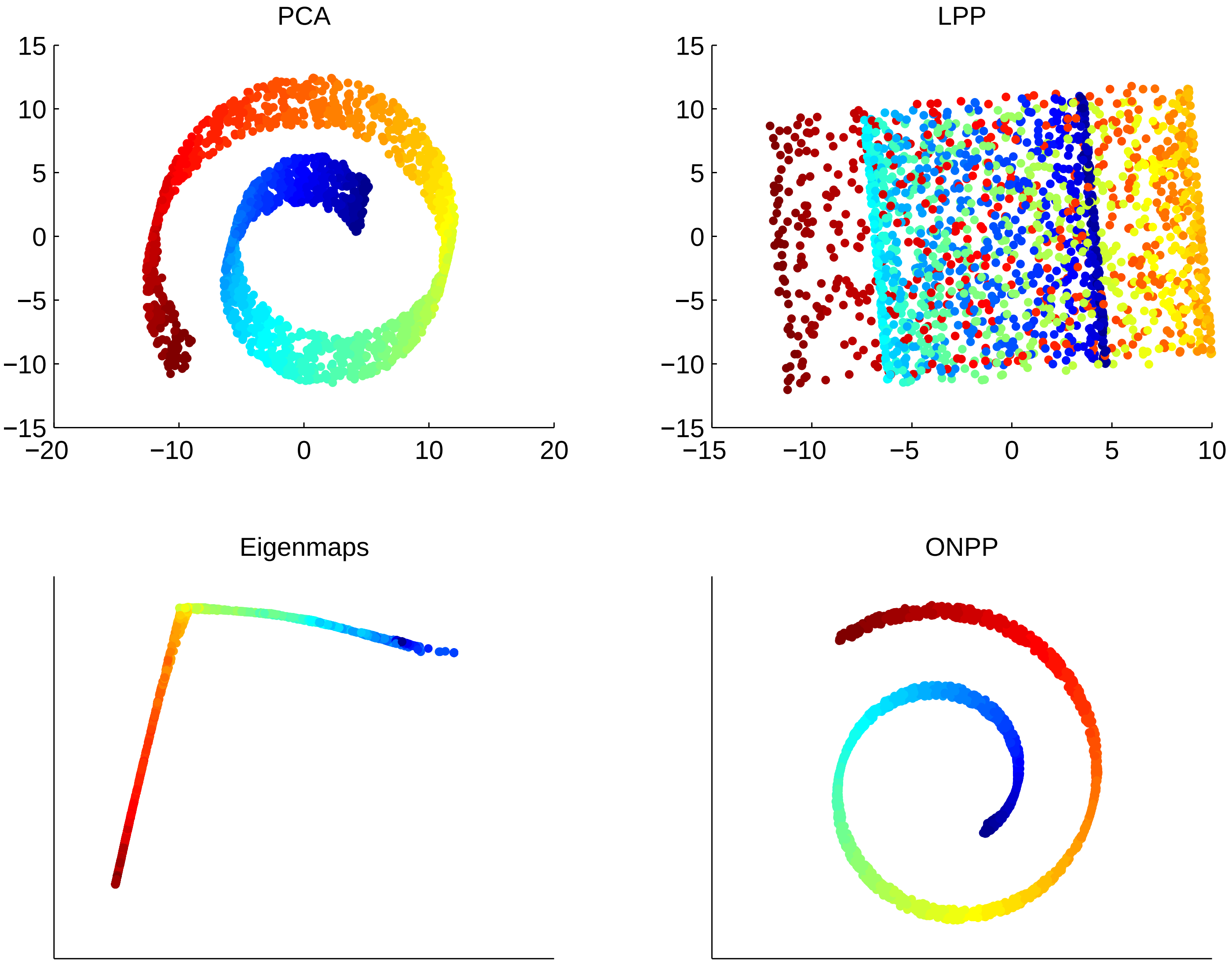
Practically: Given $X \in \mathbb{R}^{m \times n}$, we want to find a low-dimensional representation $Y \in \mathbb{R}^{d \times n}$ of X

- Two classes of methods: (1) projection techniques and (2) nonlinear implicit methods.

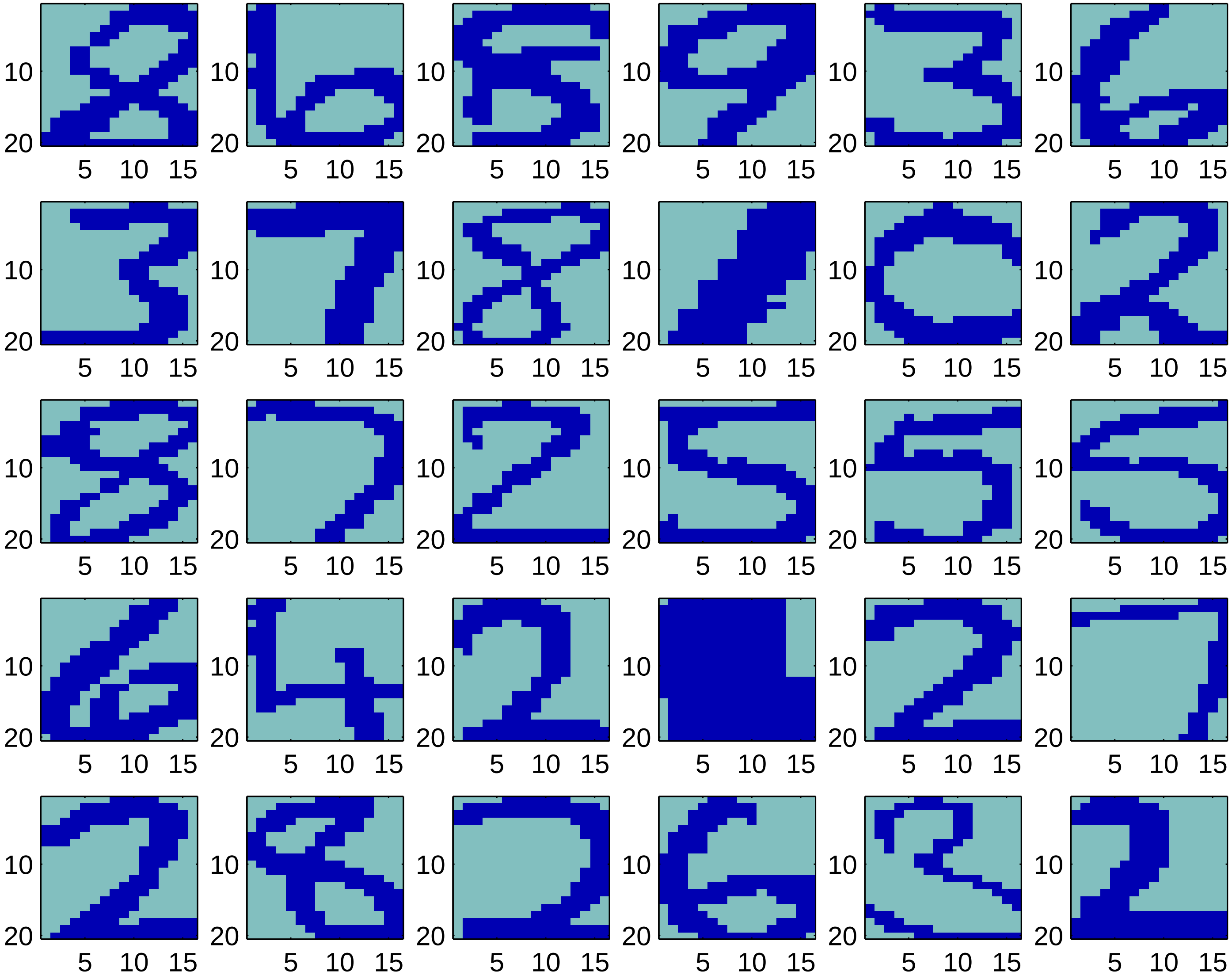
Example 1: The ‘Swill-Roll’ (2000 points in 3-D)



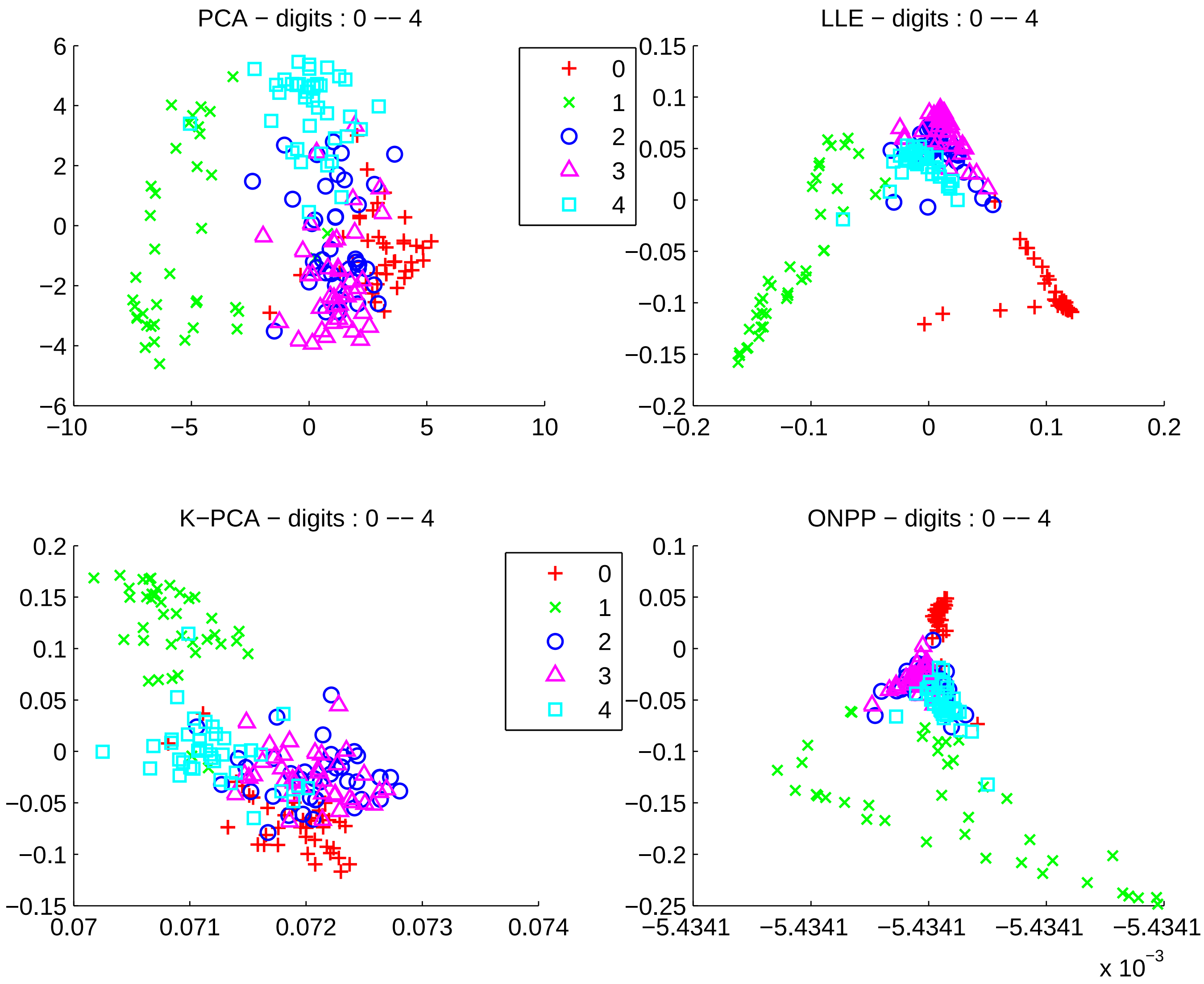
2-D 'reductions':



Example 2: Digit images (a sample of 30)



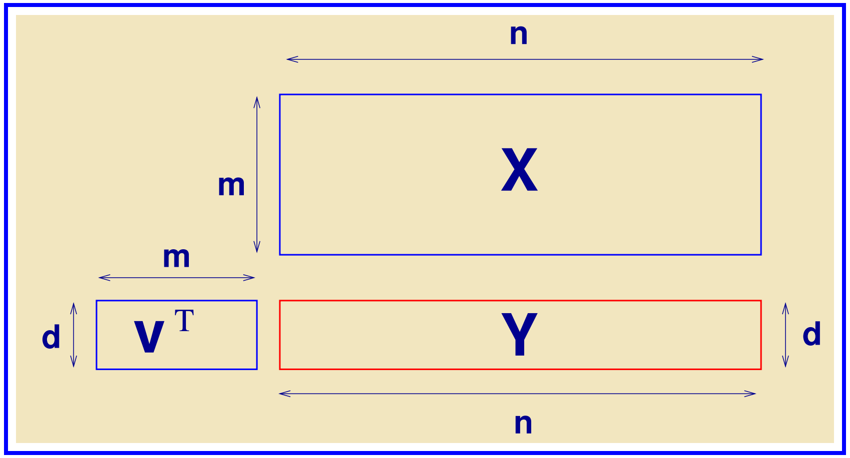
2-D 'reductions':



Projection-based Dimensionality Reduction

Given: a data set $X = [x_1, x_2, \ldots, x_n]$, and d the dimension of the desired reduced space Y.

Want: a linear transformation from X to Y



$$X \in \mathbb{R}^{m\times n}$$
$$V \in \mathbb{R}^{m\times d}$$
$$\boxed{Y = V^\top X}$$
$$\rightarrow \quad Y \in \mathbb{R}^{d\times n}$$

- m-dimens. objects (x_i) ‘flattened’ to d-dimens. space (y_i)

Constraint: The y_i’s must satisfy certain properties

- Optimization problem

Linear Dimensionality Reduction: PCA

➤ In PCA projected data must have maximum variance, i.e., we need to maximize over all orthogonal $m \times d$ matrices V:

$$\boxed{\sum_i \|y_i - \frac{1}{n}\sum_j y_j\|_2^2 = \cdots = \text{Tr}\left[V^\top \bar{X}\bar{X}^\top V\right]}$$

Where: $\bar{X} = X(I - \frac{1}{n}\mathbf{1}\mathbf{1}^T)$ == origin-recentered version of X

➤ Solution $V = \{$ dominant eigenvectors $\}$ of the covariance matrix == Set of left singular vectors of $\bar{X}$

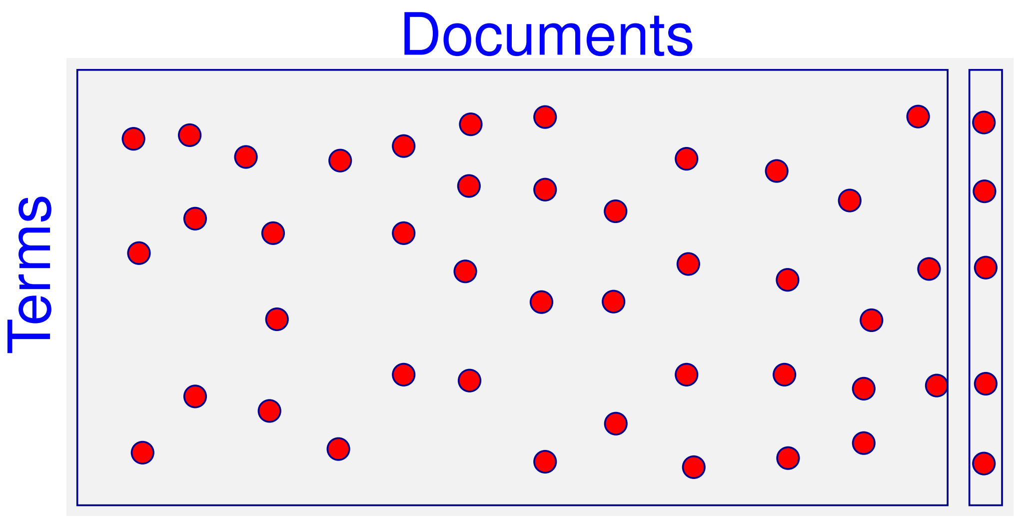
➤ Solution V also minimizes 'reconstruction error' ..

$$\sum_i \|x_i - VV^T x_i\|^2 = \sum_i \|x_i - V y_i\|^2$$

➤ .. and it also maximizes [Korel and Carmel 04] $\sum_{i,j} \|y_i - y_j\|^2$

Information Retrieval: Vector Space Model

Given: 1) set of documents (columns of a matrix A); 2) a query vector q. Entry a_{ij} of A = frequency of term i in document j + weighting.



➤ Queries ('pseudo-documents') q represented similarly to columns

Problem: find columns of A that best match q

Vector Space Model and the Truncated SVD

- Similarity metric: angle between column $A_{j,:}$ and query q

---

Use Cosines:

$$\frac{|q^T A_{:,j}|}{\|A_{:,j}\|_2 \|q\|_2}$$

- To rank all documents compute the similarity vector:

$$s = A^T q$$

- Not very effective. Problems : *polysemy*, *synonymy*, ...
- LSI: replace matrix A by low rank approximation

$$A = U\Sigma V^T \quad \rightarrow \quad A_k = U_k \Sigma_k V_k^T \quad \rightarrow \quad s_k = A_k^T q$$

- U_k : term space, V_k: document space.
- Called TSVD – Expensive, hard to update, ..

New similarity vector:

$$s_k = A_k^T q = V_k \Sigma_k U_k^T$$

Issues:

- How to select k?
- Computational cost (memory + computation)
- Problem with updates

- Alternative: SDD; Less memory but cost still an issue.
- Alternative: polynomial approximation. $s_k \approx \phi_k(A^T A) A^T q$ where ϕ_k = deg. k polynom.
- Yet another alternative: use Lanczos vectors instead of singular vectors [Ruhe and Blom, 2005]

IR: Use of the Lanczos algorithm

* Joint work with Jie Chen

➤ Lanczos is good at catching large (and small) eigenvalues: can compute singular vectors with Lanczos, & use them in LSI

➤ Can do better: Use the Lanczos vectors directly for the projection..

➤ First advocated by: K. Blom and A. Ruhe [SIMAX, vol. 26, 2005]. Use Lanczos bidiagonalization.

➤ Use a similar approach – But directly with AA^T or A^TA.

IR: Use of the Lanczos algorithm (1)

➤ Let $A \in \mathbb{R}^{m \times n}$. Apply the Lanczos procedure to $M = AA^T$. Result:

$$\boxed{Q_k^T A A^T Q_k = T_k}$$

with Q_k orthogonal, T_k tridiagonal.

➤ Define $s_i \equiv$ orth. projection of Ab on subspace span$\{Q_i\}$

$$\boxed{s_i := Q_i Q_i^T A b.}$$

➤ s_i can be easily updated from s_{i-1}:

$$s_i = s_{i-1} + q_i q_i^T A b.$$

IR: Use of the Lanczos algorithm (2)

- If $n < m$ it may be more economial to apply Lanczos to $M = A^TA$ which is $n \times n$. Result:

$$\bar{Q}_k^T A^T A \bar{Q}_k = \bar{T}_k$$

- Define:

$$t_i := A\bar{Q}_i\bar{Q}_i^T b,$$

- Project b first before applying A to result.

Why does this work?

➤ First, recall a result on Lanczos algorithm [YS 83]

Let $\{\lambda_j, u_j\}$ = j-th eigen-pair of M (label $\downarrow$)

$$\frac{\|(I - Q_k Q_k^T)u_j\|}{\|Q_k Q_k^T u_j\|} \le \frac{K_j}{T_{k-j}(\gamma_j)} \frac{\|(I - Q_1 Q_1^T)u_j\|}{\|Q_1 Q_1^T u_j\|},$$

where

$$\gamma_j = 1 + 2\frac{\lambda_j - \lambda_{j+1}}{\lambda_{j+1} - \lambda_n}, \qquad K_j = \begin{cases} 1 & j = 1 \\ \prod_{i=1}^{j-1} \frac{\lambda_i - \lambda_n}{\lambda_i - \lambda_j} & j \neq 1 \end{cases},$$

and $T_l(x)$ = Chebyshev polynomial of 1st kind of degree l.

This has the form

$$\|(I - Q_k Q_k^T)u_j\| \le c_j / T_{k-j}(\gamma_j),$$

where c_j = constant independent of k

- Result: Distance between unit eigenvector u_j and Krylov subspace span(Q_k) decays fast (for small j)
- Consider component of difference between $Ab - s_k$ along left singular directions of A. If $A = U\Sigma V^T$, then u_j's (columns of U) are eigenvectors of $M = AA^T$. So:

$$\begin{aligned} |\langle Ab - s_k, u_j\rangle| &= \left|\langle (I - Q_k Q_k^T) Ab, u_j\rangle\right| \\ &= \left|\langle (I - Q_k Q_k^T) u_j, Ab\rangle\right| \\ &\le \|(I - Q_k Q_k^T) u_j\| \|Ab\| \\ &\le c_j \|Ab\| T_{k-j}^{-1}(\gamma_j) \end{aligned}$$

- $\{s_i\}$ converges rapidly to Ab in directions of the major left singular vectors of A.

➤ Similar result for left projection sequence t_j

➤ Here is a typical distribution of eigenvalues of M: [Matrix of size $\mathbf{1398 \times 1398}$]

➤ Convergence toward first few singular vectors very fast –

Advantages of Lanczos over polynomial filters:

(1) No need for eigenvalue estimates

(2) Mat-vecs performed only in preprocessing

Disadvantages:

(1) Need to store Lanczos vectors;

(2) Preprocessing must be redone when A changes.

(3) Need for reorthogonalization – expensive for large k.

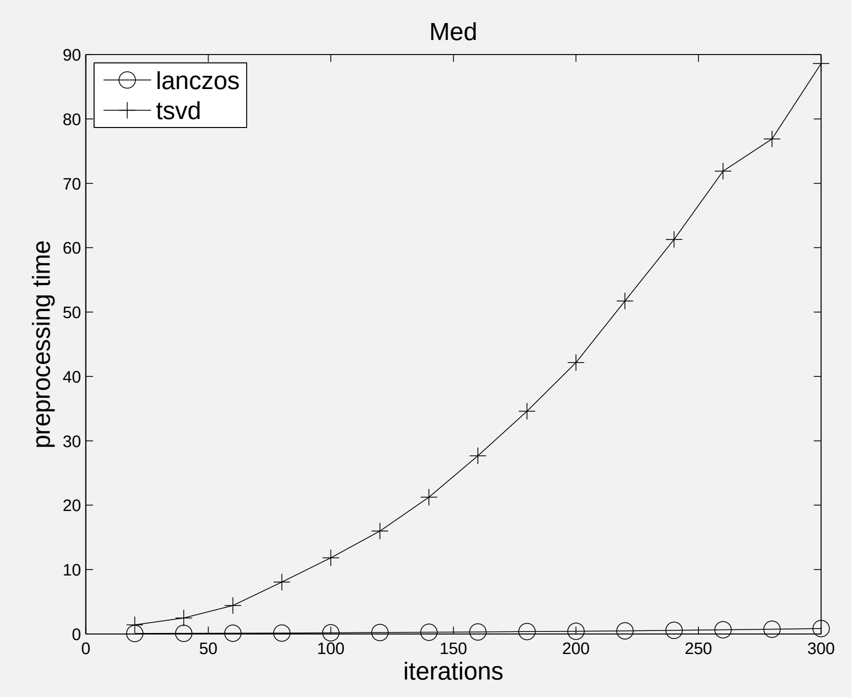
Tests: IR

Information retrieval datasets

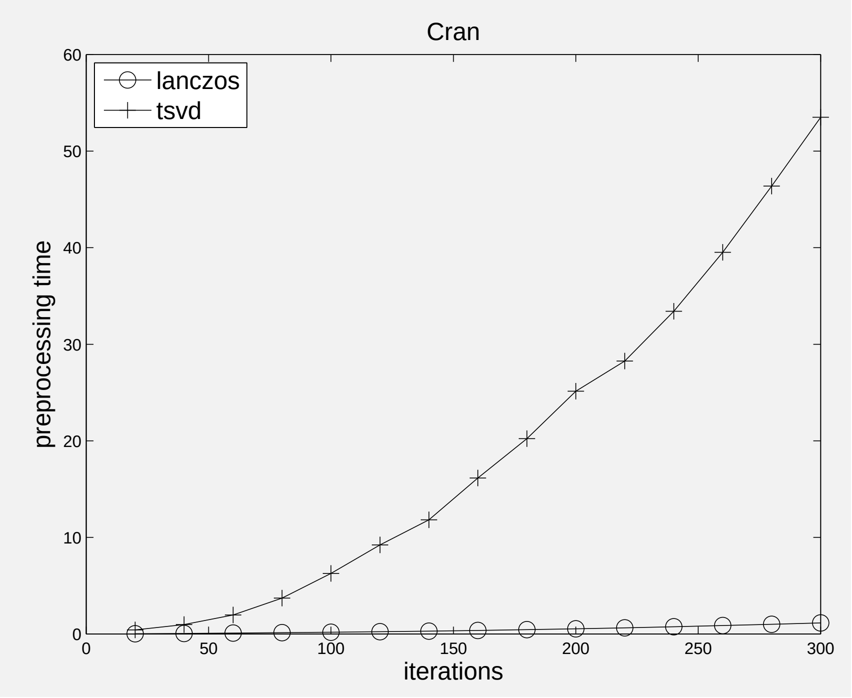
	# Terms	# Docs	# queries	sparsity
MED	7,014	1,033	30	0.735
CRAN	3,763	1,398	225	1.412

Preprocessing times

Med dataset.

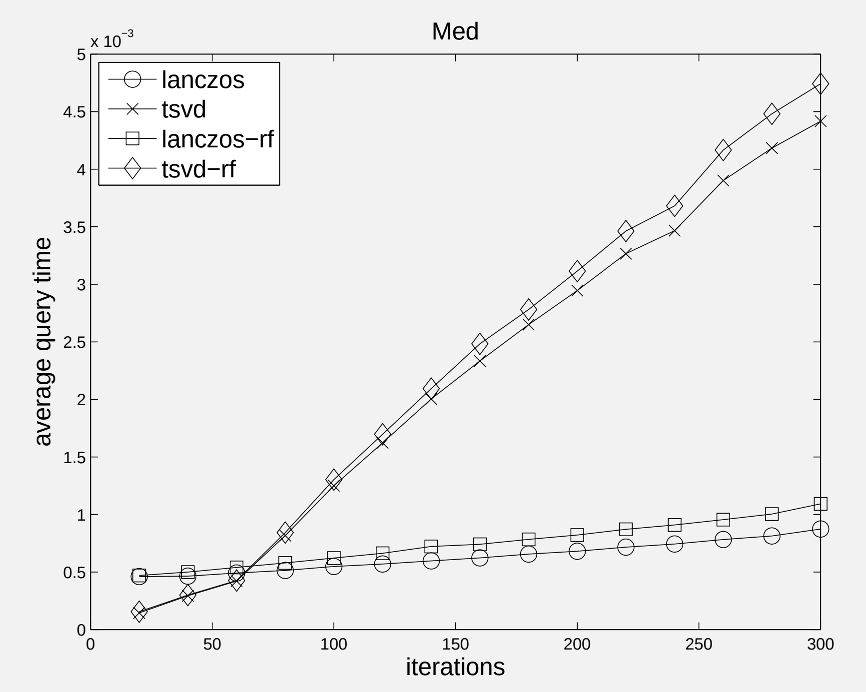


Cran dataset.

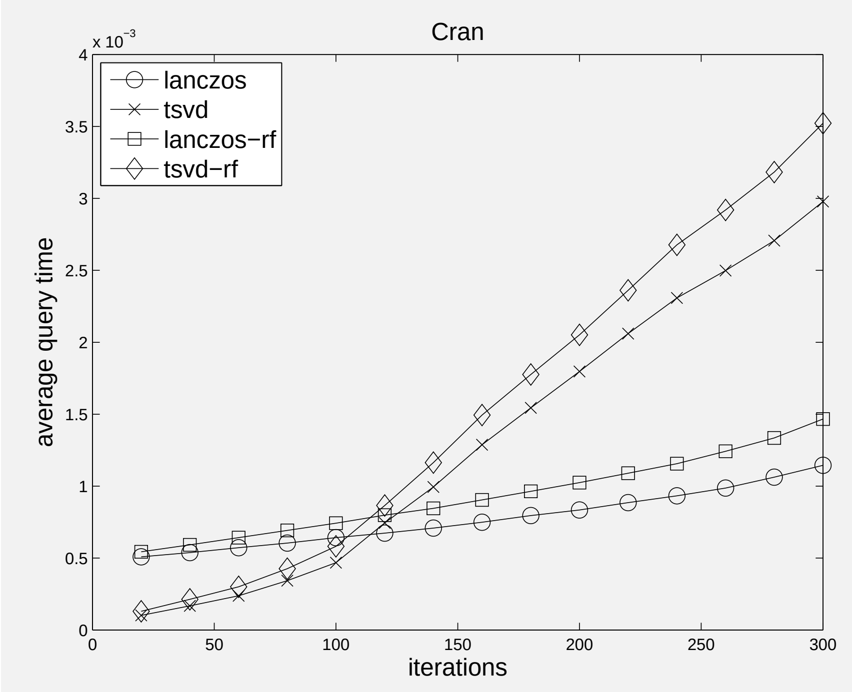


Average query times

Med dataset



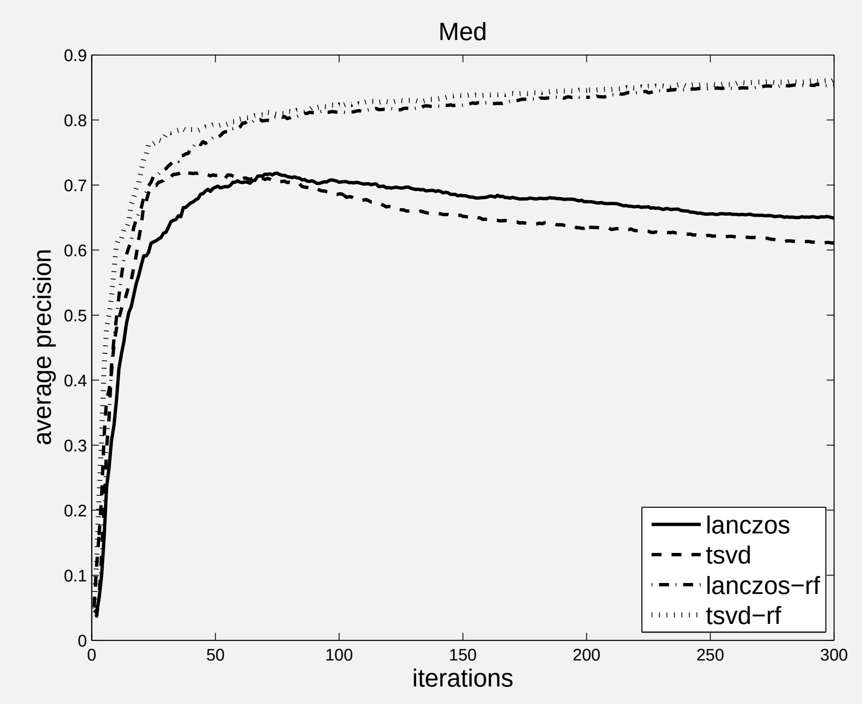
Cran dataset.

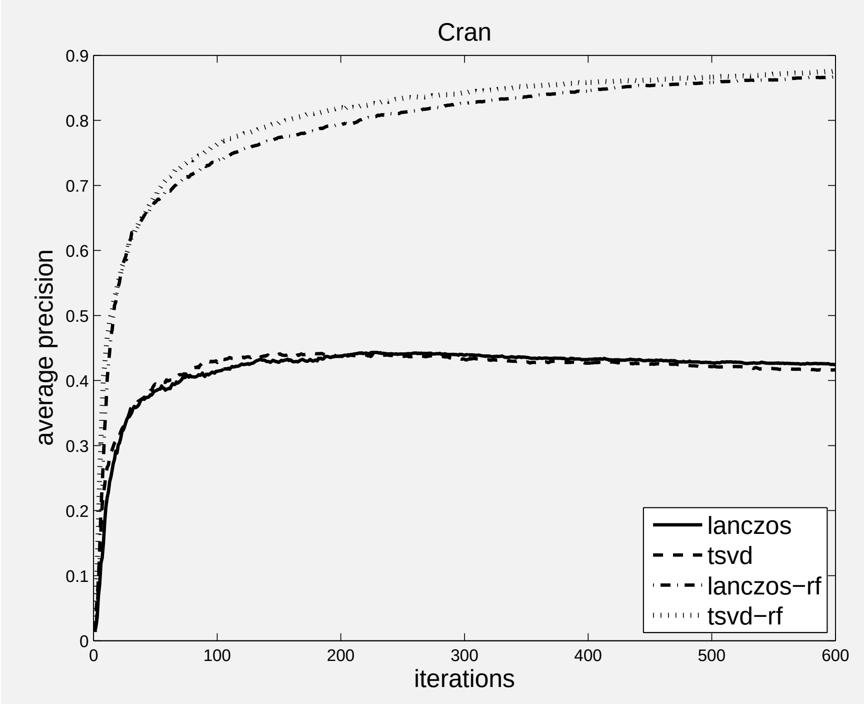


Average retrieval precision

Med dataset

Cran dataset





Retrieval precision comparisons

In summary:

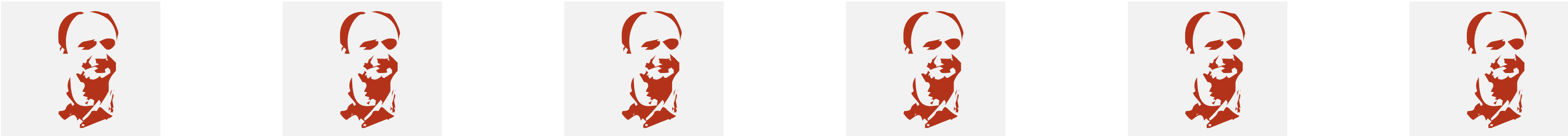
- Results comparable to those of SVD ...
- .. at a much lower cost. [However not for the same dimension k]

Thanks:

- Helpful tools and datasets widely available. We used TMG [developed at the U. of Patras (D. Zeimpekis, E. Gallopoulos)]

Problem 2: Face Recognition – background

Problem: We are given a database of images: [arrays of pixel values]. And a test (new) image.



↖ ↑ ↗

Question: Does this new image correspond to one of those in the database?

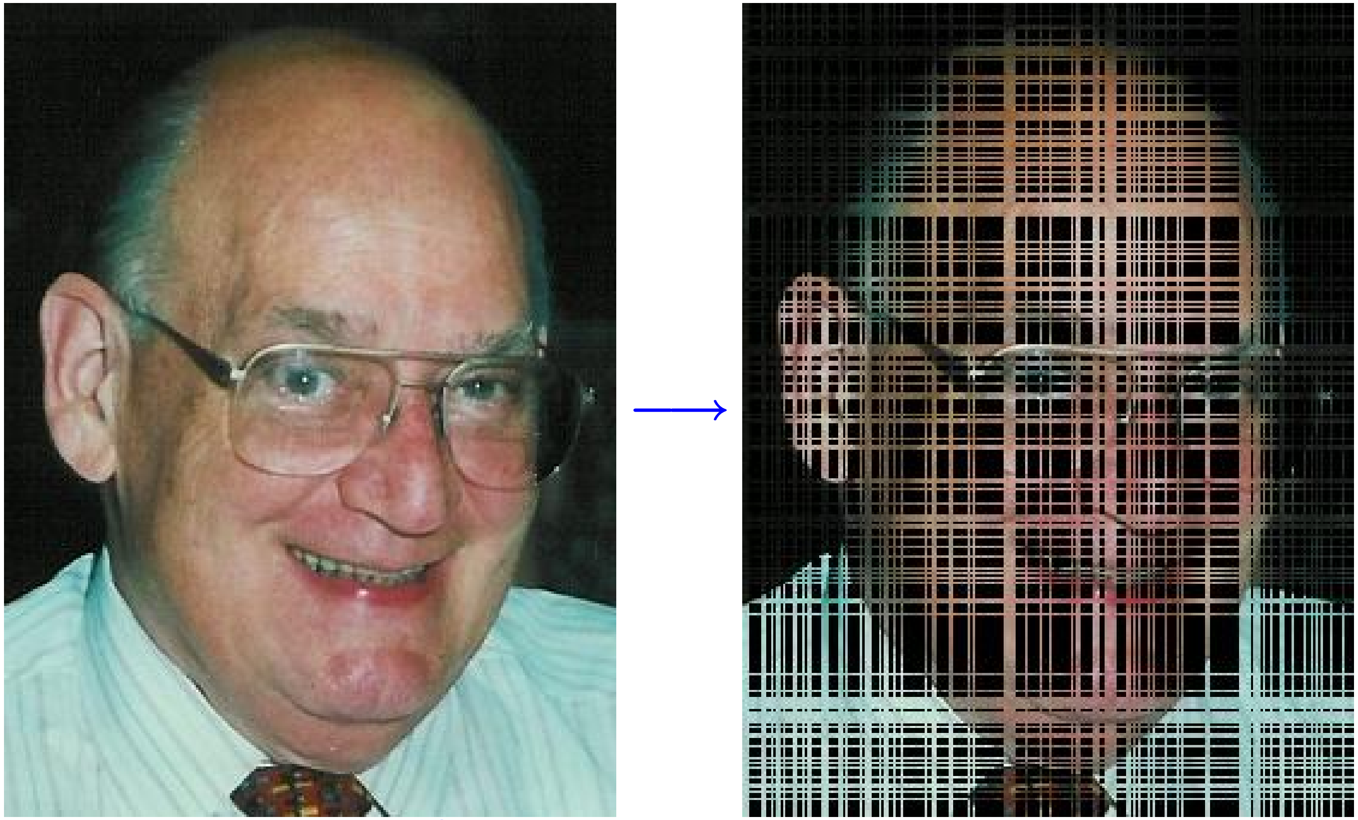
Difficulty

- Different positions, expressions, lighting, ..., situations :

Common approach: eigenfaces – Principal Component Analysis technique

Example: Occlusion. See recent paper by John Wright et al.

Right: 50% pixels randomly changed

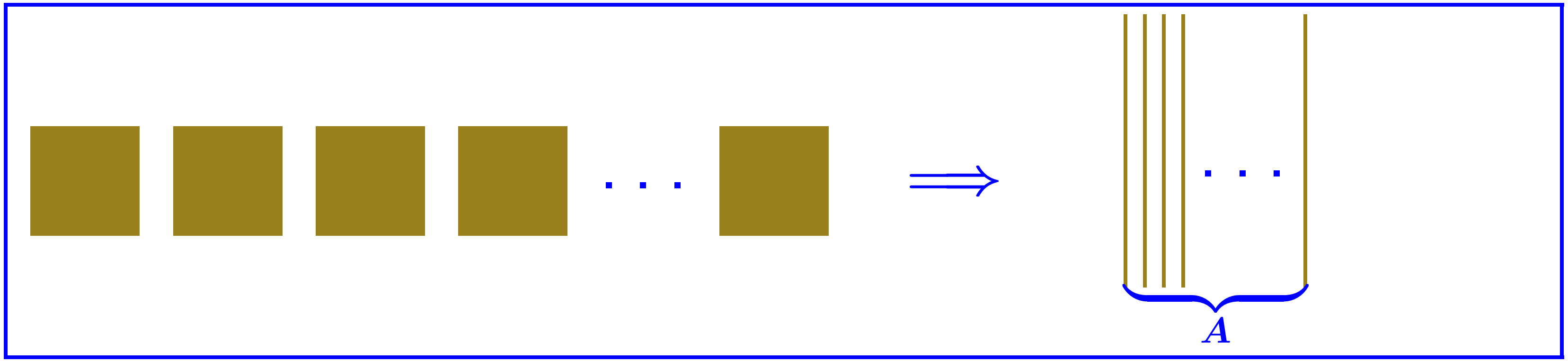


- See also: Recent real-life example – international man-hunt

Eigenfaces

– Consider each picture as a one-dimensional colum of all pixels

– Put together into an array A of size $\#\_pixels \times \#\_images$.



– Do an SVD of A and perform comparison with any test image in low-dim. space

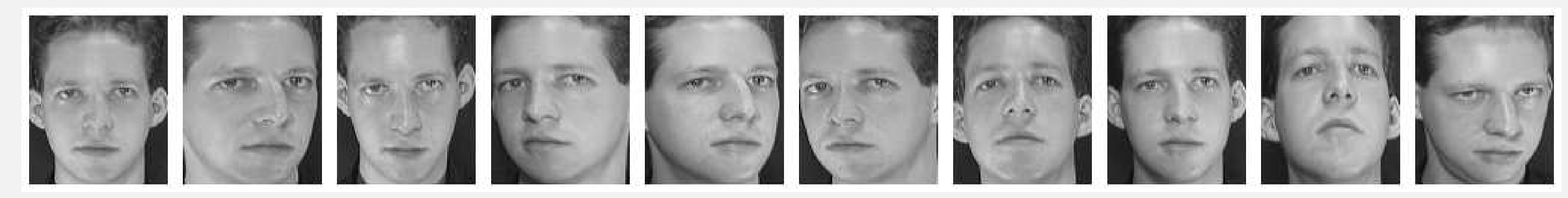
– Similar to LSI in spirit – but data is not sparse.

Idea: replace SVD by Lanczos vectors (same as for IR)

Tests: Face Recognition

Tests with 2 well-known data sets:

ORL 40 subjects, 10 sample images each – example:



# of pixels : 112×92 TOT. # images : 400

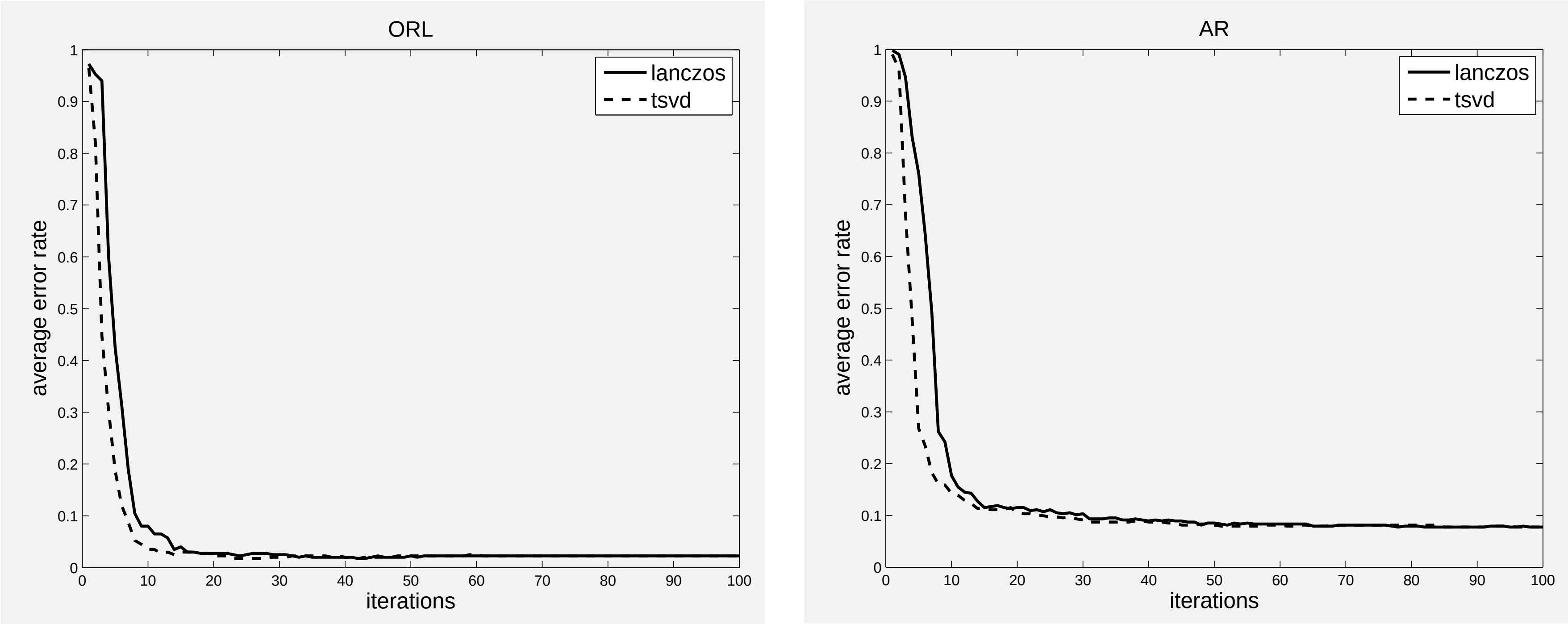
AR set 126 subjects – 4 facial expressions selected for each [natural, smiling, angry, screaming] – example:

# of pixels : 112×92 # TOT. # images : 504

Tests: Face Recognition

Recognition accuracy of Lanczos approximation vs SVD

ORL dataset　　　　　　AR dataset

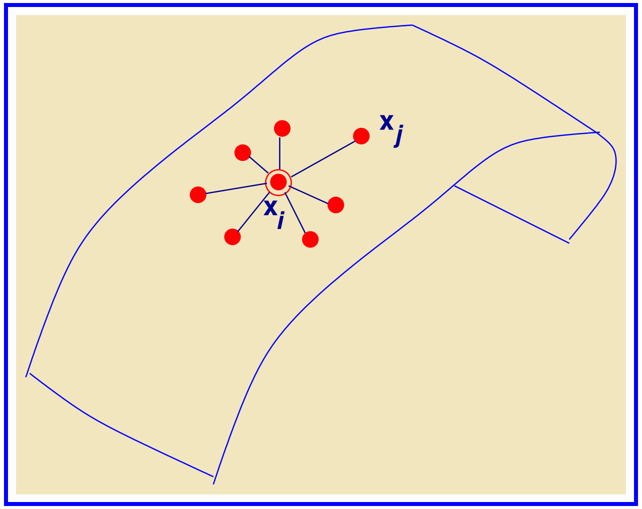


Vertical axis shows average error rate. Horizontal = Subspace dimension

GRAPH-BASED TECHNIQUES

Laplacean Eigenmaps (Belkin-Niyogi-02)

- Not a linear (projection) method but a Nonlinear method
- Starts with k-nearest-neighors graph
- Defines the graph Laplacean $L = D - W$. Simplest:



$$D = \mathrm{diag}(deg(i)); \quad w_{ij} = \begin{cases} 1 & \text{if } j \in N_i \\ 0 & \text{else} \end{cases}$$

with N_i = neighborhood of i (excl. i); $deg(i) = |N_i|$

A few properties of graph Laplacean matrices

- Let L = any matrix s.t. $L = D - W$, with $D = diag(d_i)$ and

$$w_{ij} \geq 0, \qquad d_i = \sum_{j \neq i} w_{ij}$$

Property 1: for any $x \in \mathbb{R}^n$:

$$x^\top L x = \frac{1}{2} \sum_{i,j} w_{ij} |x_i - x_j|^2$$

Property 2: (generalization) for any $Y \in \mathbb{R}^{d \times n}$:

$$\text{Tr}\,[Y L Y^\top] = \frac{1}{2} \sum_{i,j} w_{ij} \|y_i - y_j\|^2$$

Property 3: For the particular $\boldsymbol{L} = \boldsymbol{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$

$$\boldsymbol{X}\boldsymbol{L}\boldsymbol{X}^\top = \bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^\top == n \times \text{Covariance matrix}$$

[Proof: 1) $\boldsymbol{L}$ is a projector: $\boldsymbol{L}^\top\boldsymbol{L} = \boldsymbol{L}^2 = \boldsymbol{L}$, and 2) $\boldsymbol{X}\boldsymbol{L} = \bar{\boldsymbol{X}}$]

- Consequence-1: PCA equivalent to maximizing $\sum_{ij} \|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2$
- Consequence-2: what about replacing trivial $\boldsymbol{L}$ with something else? [viewpoint in Koren-Carmel'04]

Property 4: (Graph partitioning) If x is a vector of signs (± 1) then

$$x^\top L x = 4 \times \text{('number of edge cuts')}$$

edge-cut = pair (i, j) with $x_i \neq x_j$

➤ Consequence: Can be used for partitioning graphs, or 'clustering'

[take $p = sign(u_2)$, where u_2 = 2nd smallest eigenvector..]

Return to Laplacean eigenmaps approach

Laplacean Eigenmaps *minimizes*

$$\mathcal{F}_{EM}(Y) = \sum_{i,j=1}^{n} w_{ij} \|y_i - y_j\|^2 \quad \text{subject to} \quad YDY^\top = I \ .$$

Notes:

1. Motivation: if $\|x_i - x_j\|$ is small (orig. data), we want $\|y_i - y_j\|$ to be also small (low-D data)

2. Note Min instead of Max as in PCA [counter-intuitive]

3. Above problem uses original data indirectly through its graph

- Problem translates to:

$$\min_{\begin{cases} Y \in \mathbb{R}^{d\times n} \\ Y D\, Y^\top = I \end{cases}} \text{Tr}\left[Y(D-W)Y^\top\right] .$$

- Solution (sort eigenvalues increasingly):

$$(D-W)u_i = \lambda_i D u_i \; ; \quad y_i = u_i^\top ; \quad i = 1, \cdots, d$$

- Note: an $n \times n$ sparse eigenvalue problem [In 'sample' space]
- Note: can assume $D = I$. Amounts to rescaling data. Problem becomes

$$\boxed{(I-W)u_i = \lambda_i u_i \; ; \quad y_i = u_i^\top ; \quad i = 1, \cdots, d}$$

Why smallest eigenvalues vs largest for PCA?

Intuition:

Graph Laplacean and 'unit' Laplacean are very different: one involves a sparse graph (More like a discr. differential operator). The other involves a dense graph. (More like a discr. integral operator). They should be treated as the inverses of each other.

➤ Viewpoint confirmed by what we learn from Kernel approach

Locally Linear Embedding (Roweis-Saul-00)

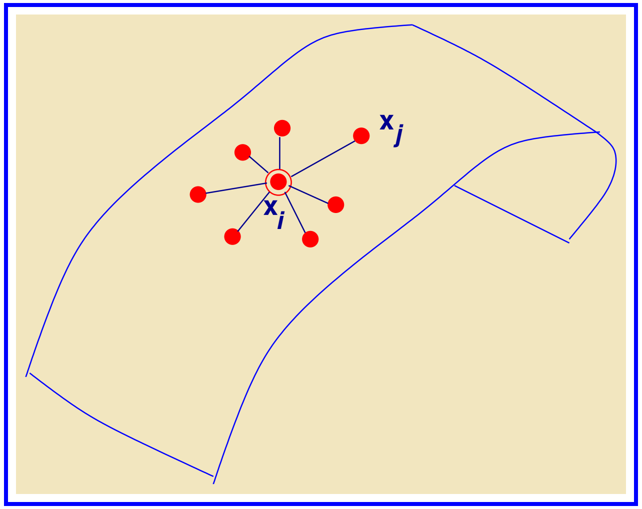
➤ LLE is very similar to Eigenmaps. Main differences:

1) Graph Laplacean matrix is replaced by an 'affinity' graph

2) Objective function is changed: want to preserve graph

1. Graph: Each x_i is written as a convex combination of its k nearest neighbors:

$x_i \approx \Sigma w_{ij} x_j, \quad \sum_{j \in N_i} w_{ij} = 1$



➤ Optimal weights computed ('local calculation') by minimizing

$$\|x_i - \Sigma w_{ij} x_j\| \quad \text{for} \quad i = 1, \cdots, n$$

2. Mapping:

The y_i's should obey the same 'affinity' as x_i's $\rightsquigarrow$

Minimize:

$$\sum_i \left\| y_i - \sum_j w_{ij} y_j \right\|^2 \quad \text{subject to:} \quad Y\mathbf{1} = 0, \quad YY^\top = I$$

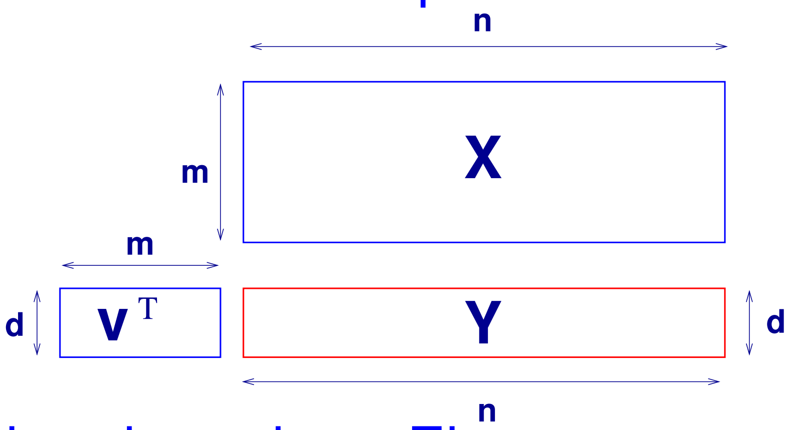
Solution:

$$\boxed{(I - W^\top)(I - W)u_i = \lambda_i u_i; \qquad y_i = u_i^\top.}$$

- $(I - W^\top)(I - W)$ replaces the graph Laplacean of eigenmaps

Locally Preserving Projections (He-Niyogi-03)

- LPP is a linear dimensionality reduction technique



- Recall the setting:

Want $V \in \mathbb{R}^{m \times d}$; $Y = V^\top X$

- Starts with the same neighborhood graph as Eigenmaps: $L \equiv D - W$ = graph 'Laplacean'; with $D \equiv diag(\{\Sigma_i w_{ij}\})$.

➤ Optimization problem is to solve

$$\min_{\boldsymbol{Y} \in \mathbb{R}^{d\times n},\ \boldsymbol{Y}\boldsymbol{D}\boldsymbol{Y}^\top = \boldsymbol{I}} \quad \Sigma_{i,j} w_{ij} \left\| \boldsymbol{y}_i - \boldsymbol{y}_j \right\|^2, \quad \boldsymbol{Y} = \boldsymbol{V}^\top \boldsymbol{X}.$$

➤ Difference with eigenmaps: $\boldsymbol{Y}$ is a projection of $\boldsymbol{X}$ data

➤ Solution (sort eigenvalues increasingly)

$$\boldsymbol{X}\boldsymbol{L}\boldsymbol{X}^\top \boldsymbol{v}_i = \lambda_i \boldsymbol{X}\boldsymbol{D}\boldsymbol{X}^\top \boldsymbol{v}_i \quad \boldsymbol{y}_{i,:} = \boldsymbol{v}_i^\top \boldsymbol{X}$$

➤ Note: essentially same method in [Koren-Carmel'04] called 'weighted PCA' [viewed from the angle of improving PCA]

ONPP (Kokiopoulou and YS '05)

- Orthogonal Neighborhood Preserving Projections
- Can be viewed as a linear version of LLE
- Uses the same graph as LLE. Objective: preserve the affinity graph (as in LEE) *but* by means of an orthogonal projection
- Objective function

$$\boldsymbol{\Phi(Y)} = \Sigma_i \|\boldsymbol{y}_i - \Sigma_j w_{ij} \boldsymbol{y}_j\|^2 \quad \text{Constraint: } \boldsymbol{Y} = \boldsymbol{V}^\top \boldsymbol{X},\ \boldsymbol{V}^\top \boldsymbol{V} = \boldsymbol{I}$$

- Notice that

$$\boldsymbol{\Phi(Y)} = \|\boldsymbol{Y} - \boldsymbol{Y}\boldsymbol{W}^\top\|_F^2 = \cdots = \text{Tr}\left[\boldsymbol{V}^\top \boldsymbol{X}(\boldsymbol{I} - \boldsymbol{W}^\top)(\boldsymbol{I} - \boldsymbol{W})\boldsymbol{X}^\top \boldsymbol{V}\right]$$

Resulting problem:

$$\min_{\substack{V \in \mathbb{R}^{m\times d};\\ V^\top V = I}} \mathbf{Tr}\left[V^\top \underbrace{X(I - W^\top)(I - W)X^\top}_{M} V \right]$$

Solution: Columns of V = eigenvectors of M associated with smallest d eigenvalues

➤ Can be computed as d lowest left singular vectors of

$$X(I - W^\top)$$

A unified view

Method	Object. (min)	Constraint
PCA/MDS	Tr $[V^\top X(-I + ee^\top)X^\top V]$	$V^\top V = I$
LLE	Tr $[Y(I - W^\top)(I - W)Y^\top]$	$YY^\top = I$
Eigenmaps	Tr $[Y(I - W)Y^\top]$	$YY^\top = I$
LPP	Tr $[V^\top X(I - W)X^\top V]$	$V^\top XX^\top V = I$
ONPP	Tr $[V^\top X(I - W^\top)(I - W)X^\top V]$	$V^\top V = I$
LDA	Tr $[V^\top X(I - H)X^\top V]$	$V^\top XX^\top V = I$

- Let $M = I - W$ = a Laplacean matrix ($-I + ee^\top$ for PCA/MDS); or the LLE matrix $(I - W)(I - W^\top)$, or geodesic distance matrix (ISOMAP).
- All techniques lead to one of two types of problems
- First type is:
$$\boxed{\min_{\begin{cases} Y \in \mathbb{R}^{d\times n} \\ YY^\top = I \end{cases}} \textbf{Tr}\left[YMY^\top\right]}$$
- Y obtained from solving an eigenvalue problem
- LLE, Eigenmaps (normalized), ..

➤ And the second type is:

$$\begin{cases} \min\limits_{V \in \mathbb{R}^{m \times d}} & \textbf{Tr}\left[V^\top X M X^\top V\right] \\ V^\top G\, V = I \end{cases}$$

➤ G is either the identity matrix or $XDX^\top$ or $XX^\top$.

➤ Low-Dim. data : $Y = V^\top X$

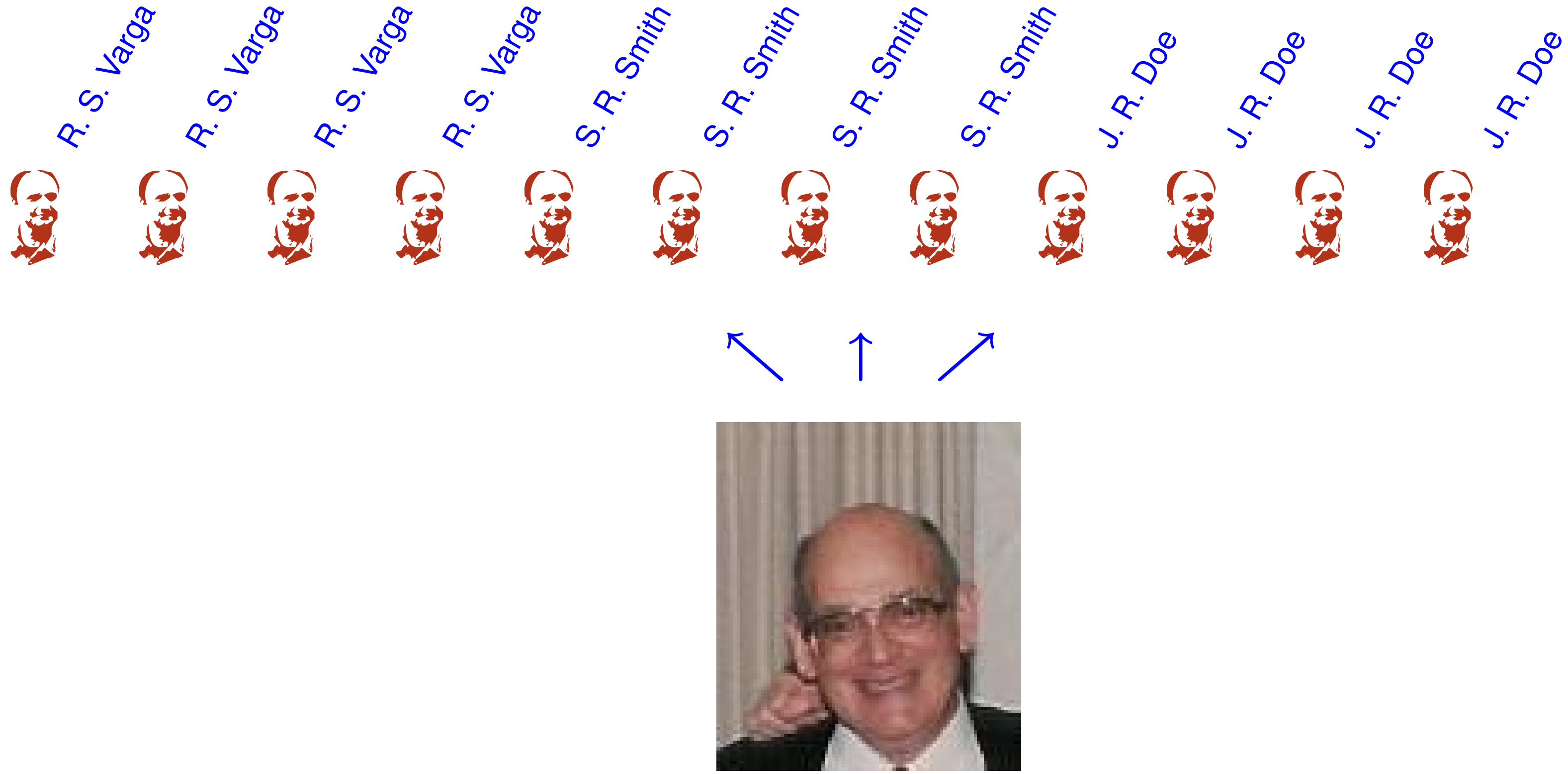
Important observation: 2nd is just a projected version of the 1st, i.e., approximate eigenvectors are sought in Span $\{X\}$ [Rayleigh-Ritz procedure]

➤ Problem is of dim. m (dim. of data) not n (# of samples).

➤ This difference can be mitigated by resorting to Kernels..

Graph-based methods in a supervised setting

➤ Subjects of training set are known (labeled). Q: given a test image (say) find its label.



Question: Find label (best match) for test image.

Methods can be adapted to supervised mode by building the graph to take into account class labels. Idea is simple: Build G so that nodes in the same class are neighbors. If c = # classes, G will consist of c cliques.

- Matrix W will be block-diagonal

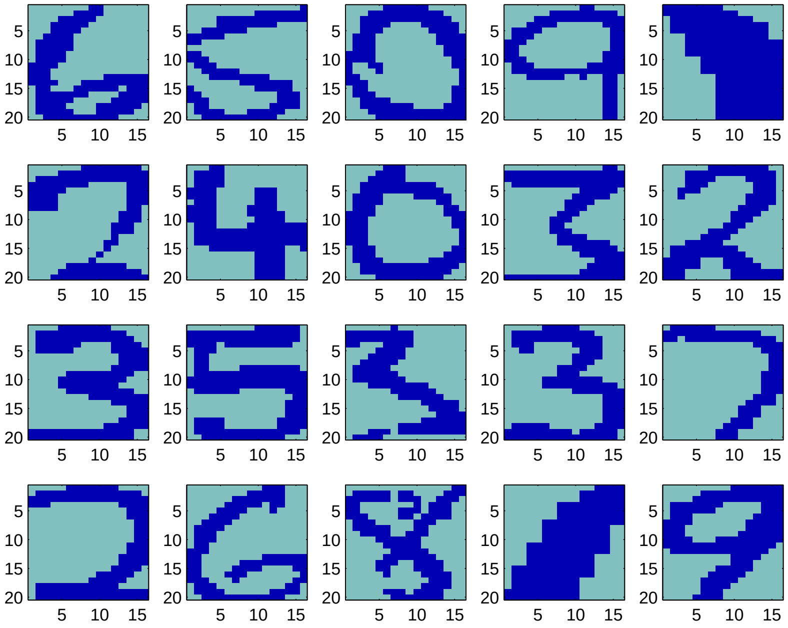
$$W = diag(W_1, W_2, \cdots, W_c)$$

- Easy to see that $\mathrm{rank}(W) = n - c$.
- Can be used for LPP, ONPP, etc..

TIME FOR A MATLAB DEMO

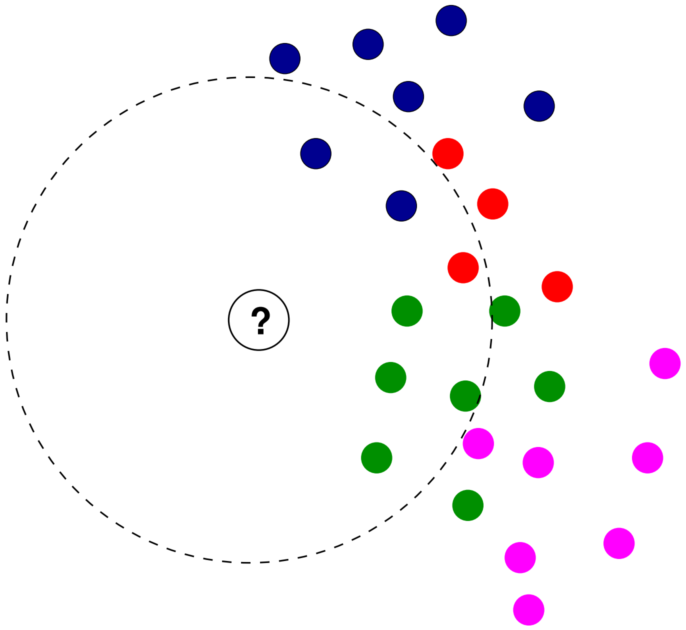
Supervised learning experiments: digit recognition

- Set of 390 images of digits (39 of each digit)
- Each picture has $20 \times 16 = 320$ pixels.
- Select any one of the digits and try to recognize it among the 389 remaining images
- Methods: KNN, PCA, LPP, ONPP



One word about KNN-classifiers

- Simple idea of ’vote’ – get k nearest neighbors.
- Assigned label = ‘most common label among these neibhbors’



MULTILEVEL METHODS

Multilevel techniques

- Divide and conquer paradigms as well as multilevel methods in the sense of ‘domain decomposition’
- Main principle: very costly to do an SVD [or Lanczos] on the whole set. Why not find a smaller set on which to do the analysis – without too much loss?
- Tools used: graph coarsening, divide and conquer –

Multilevel techniques: Hypergraphs to the rescue

General idea: Given $X = [x_1, \cdots, x_n] \in \mathbb{R}^{m \times n}$ find another set ('coarsened set') $\hat{X} = [\hat{x}_1, \cdots, \hat{x}_k] \in \mathbb{R}^{m \times k}$

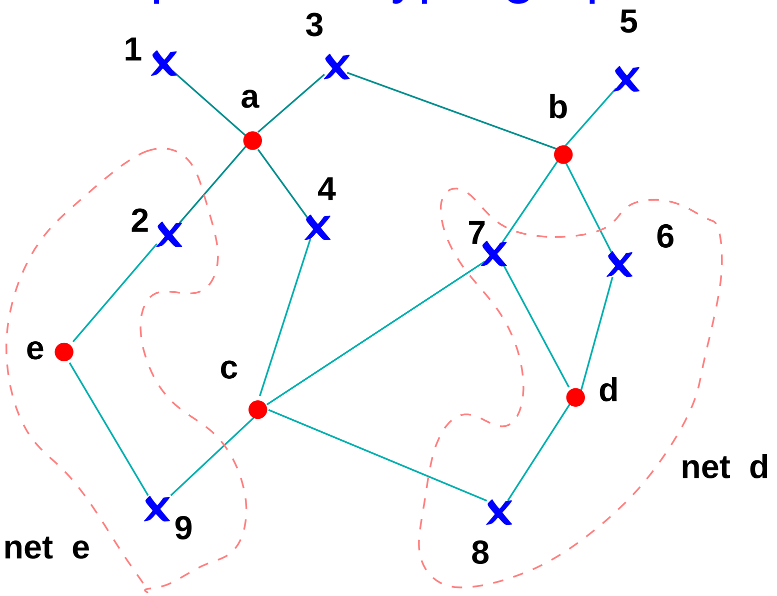
- $\hat{X}$ should be a good representative of X –
- then find projector from $\mathbb{R}^m$ to $\mathbb{R}^d$ based on this subset
- Main tool used: graph coarsening.
- We will describe hypergraph-based techniques

Hypergraphs

A hypergraph $H = (V, E)$ is a generalizaition of a graph

- V = set of vertices V
- E = set of hyperedges. Each $e \in E$ is a nonempty subset of V
- Standard undirected graphs: each e consists of two vertices.
- *degree* of e = $|e|$
- *degree* of a vertex v = number of hyperedges e s.t. $x \in e$.
- Two vertices are *neighbors* if there is a hyperedge connecting them

Example of a hypergraph



Boolean matrix representation

$$A = \begin{array}{ccccccccc|c}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & \\
\hline
1 & 1 & 1 & 1 & & & & & & a \\
 & & 1 & & 1 & 1 & 1 & & & b \\
 & & & 1 & & & 1 & 1 & 1 & c \\
 & & & & & 1 & 1 & 1 & & d \\
 & 1 & & & & & & & 1 & e
\end{array}$$

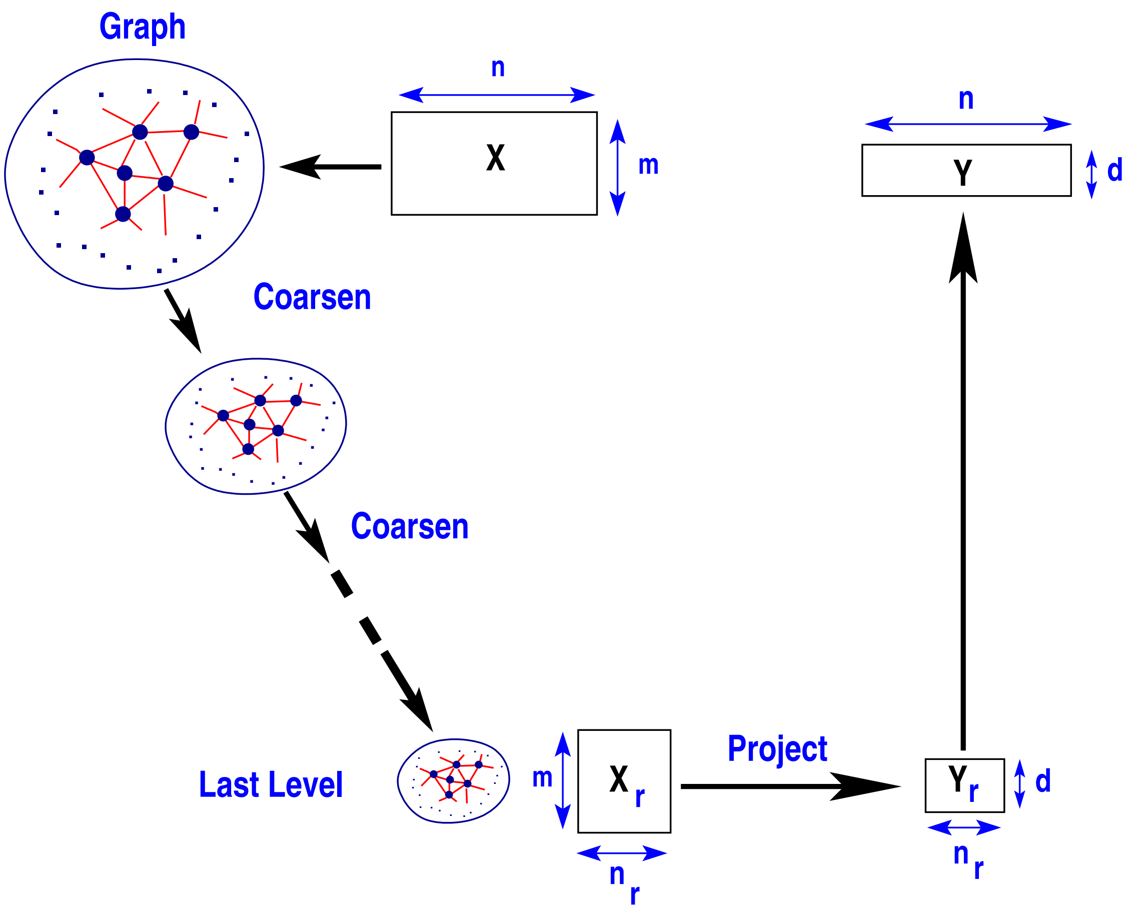
- Canonical hypergraph representation for sparse data (e.g. text)

Hypergraph Coarsening

- Coarsening a hypergraph $H = (V, E)$ means finding a 'coarse' approximation $\hat{H} = (\hat{V}, \hat{E})$ to H with $|\hat{V}| < |V|$, which tries to retain as much as possible of the structure of the original hypergraph
- Idea: repeat coarsening recursively.
- Result: succession of smaller hypergraphs which approximate the original graph.
- Several methods exist. We use 'matching', which successively merges pairs of vertices
- Used in most graph partitioning methods: hMETIS, Patoh, zoltan, ..
- Algorithm successively selects pairs of vertices to merge – based on measure of similarity of the vertices.

Application: Multilevel Dimension Reduction

Main Idea: coarsen to a certain level. Then use the resulting data set $\hat{X}$ to find projector from $\mathbb{R}^m$ to $\mathbb{R}^d$. This projector can be used to project the original data or any new data.



➤ Main gain: Dimension reduction is done with a much smaller set.
Hope: not much loss compared to using whole data

Application to text mining

- Recall common approach:

1. Scale data [e.g. TF-IDF scaling:]
2. Perform a (partial) SVD on resulting matrix $X \approx U_d \Sigma_d V_d^T$
3. Process query by same scaling (e.g. TF-IDF)
4. Compute similarities in d-dimensional space: $s_i = \langle \hat{q}, \hat{x}_i \rangle / \|\hat{q}\| \|\hat{x}_i\|$

where $[\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n] = V_d^T \in \mathbb{R}^{d \times n}$; $\quad \hat{q} = \Sigma_d^{-1} U_d^T \bar{q} \in \mathbb{R}^d$

- Multilevel approach: replace SVD (or any other dim. reduction) by dimension reduction on coarse set. Only difference: TF-IDF done on the coarse set not original set.

Tests

Three public data sets used for experiments: `Medline`, `Cran` and `NPL` (cs.cornell.edu)

- Coarsening to a max. of 4 levels.

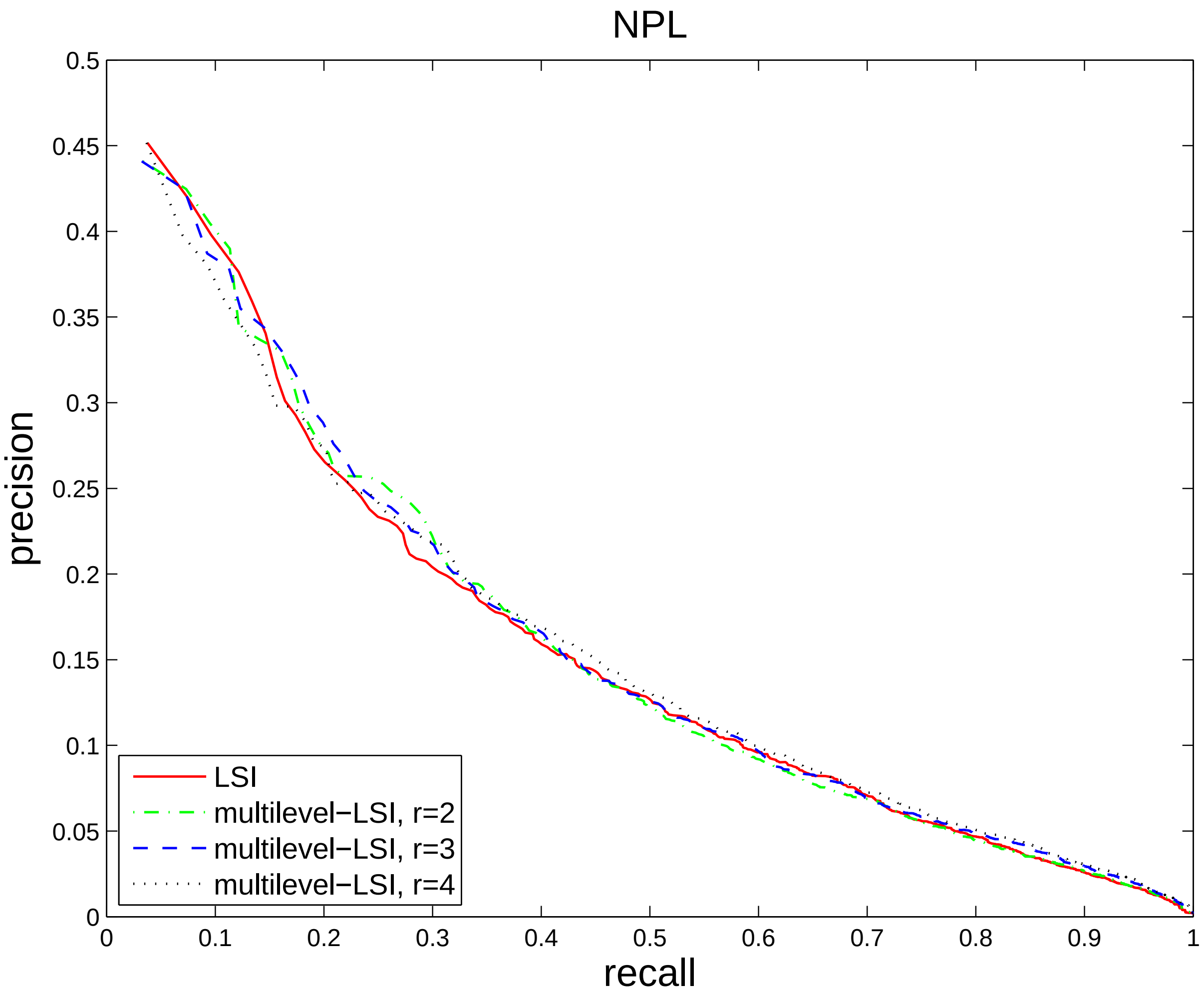
Data set	`Medline`	`Cran`	`NPL`
# documents	1033	1398	11429
# terms	7014	3763	7491
sparsity (%)	0.74%	1.41%	0.27%
# queries	30	225	93
avg. # rel./query	23.2	8.2	22.4

Results with NPL

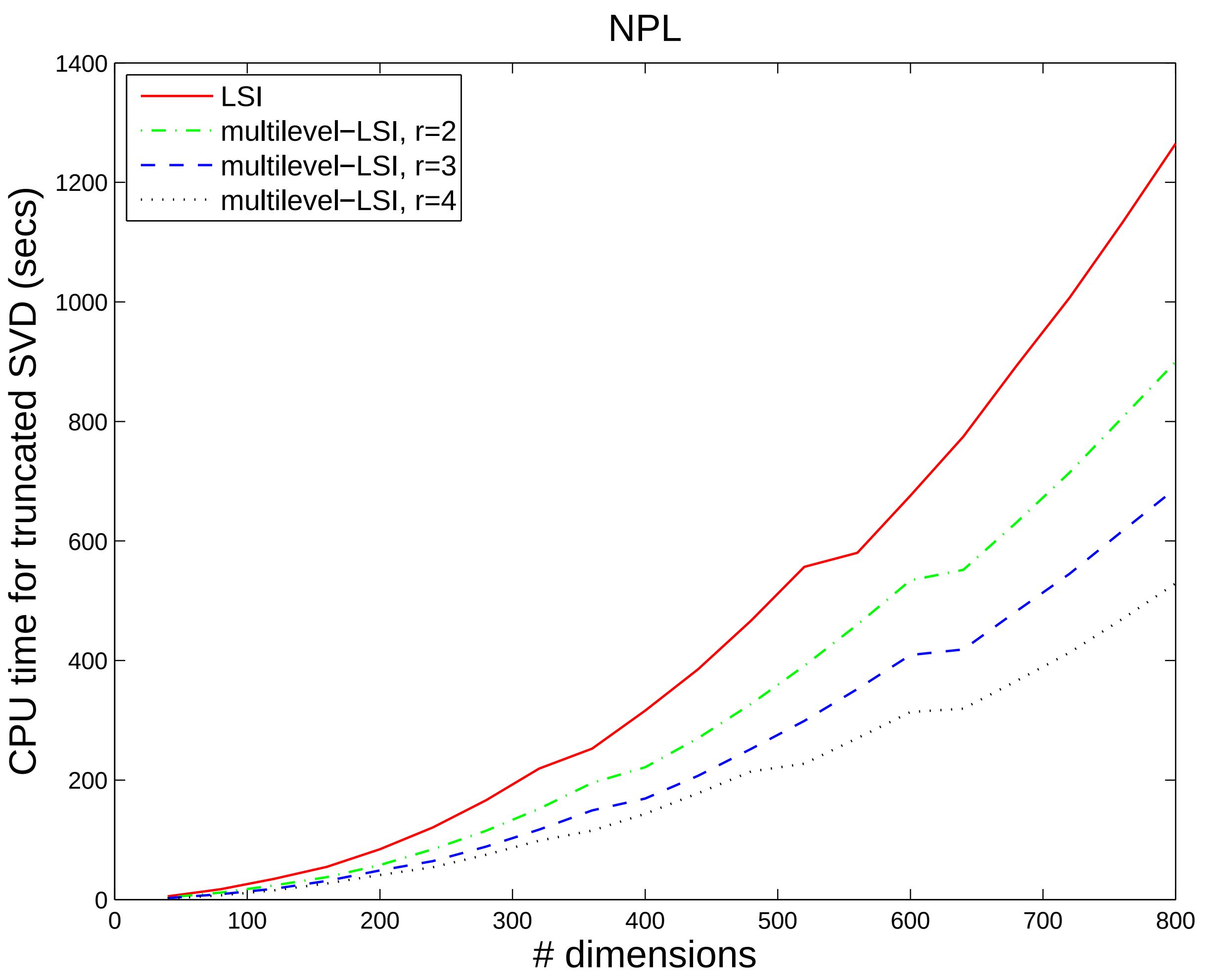
Statistics

Level	coarsen. time	# doc.	optimal # dim.	optimal avg. precision
#1	N/A	11429	736	23.5%
#2	3.68	5717	592	23.8%
#3	2.19	2861	516	23.9%
#4	1.50	1434	533	23.3%

Precision-Recall curves



CPU times for preprocessing (Dim. reduction part)



Conclusion

➤ So how is this related to intitial title of "efficient algorithms in data mining"?

➤ Answer: All these eigenvalue problems are not cheap to solve..

➤ .. and cost issue does not seem to bother practitioners too much for now..

➤ Ingredients that will become mandatory:

1. Avoid the SVD
2. Fast algorithms that do not sacrifice quality.
3. In particullar: Multilevel approaches
4. Multilinear algebra [tensors]