Behavioral Data Mining

Lecture 15
Factor Models

Outline

- Motivation
- Linear subspaces: SVD and PCA
- LSI
- Netflix/Alternating Least Squares
- Non-negative Matrix Factorization
- Topic Models: LDA

Motivation

- For problems with linear relationships between a response *Y* and feature data *X* we have the linear regression formula:

$$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^{p} X_j \hat{\beta}_j$$

- But as the dimension of *Y* grows, we quickly "run out of data" – the model has more degrees of freedom than we have data.
  - e.g. recommendations, information retrieval
  - Features and responses are the same, e.g. ratings of products, or relevance of a term to a document.
  - We have to deal with thousands or millions of responses, as well as thousands or millions of features.

- This is one flavor of the "curse of dimensionality".

Motivation

- High-dimensional data will often have simpler structure, e.g.
  - Topics in documents
  - General product characteristics (reliability, cost etc.)
  - User preferences
  - User personality, demographics
  - Dimensions of sentiment
  - Themes in discussions
  - "Voices" or styles in documents
- We can approximate these effects using a lower-dimensional model, which we assume for now is linear.

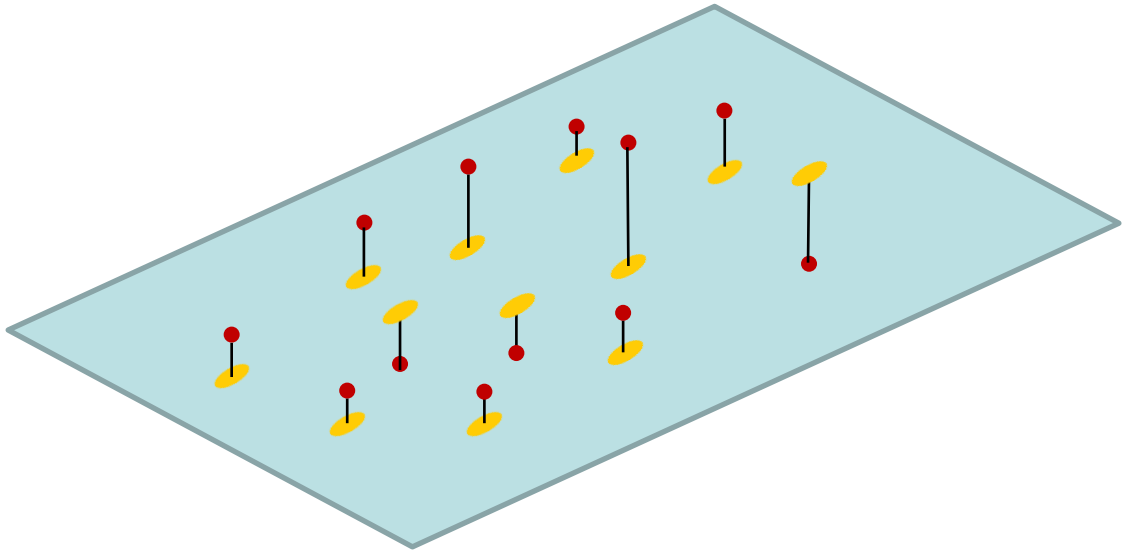
Motivation

The subspace allows us to predict the values of other features from known features:

d coordinates define a unique point on a d-dimensional plane.

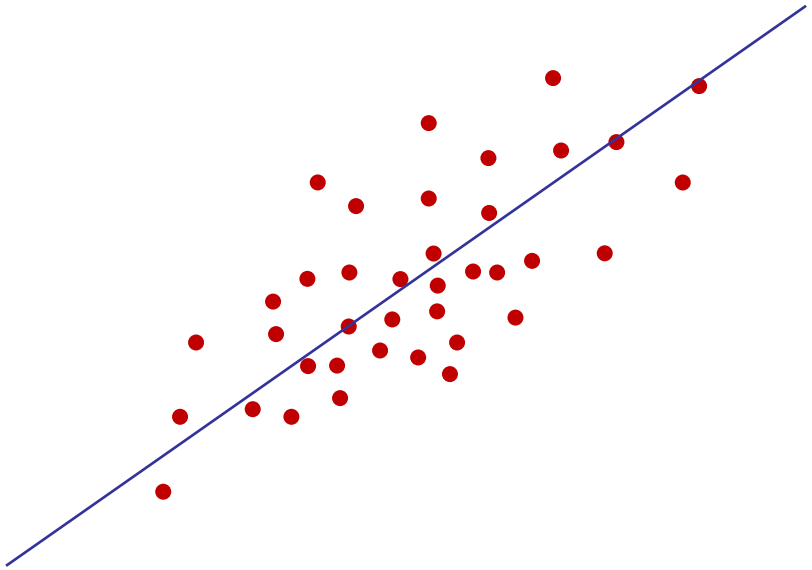
From these measurements, we can predict other coordinates.

Applications: Rating/Review/Survey prediction



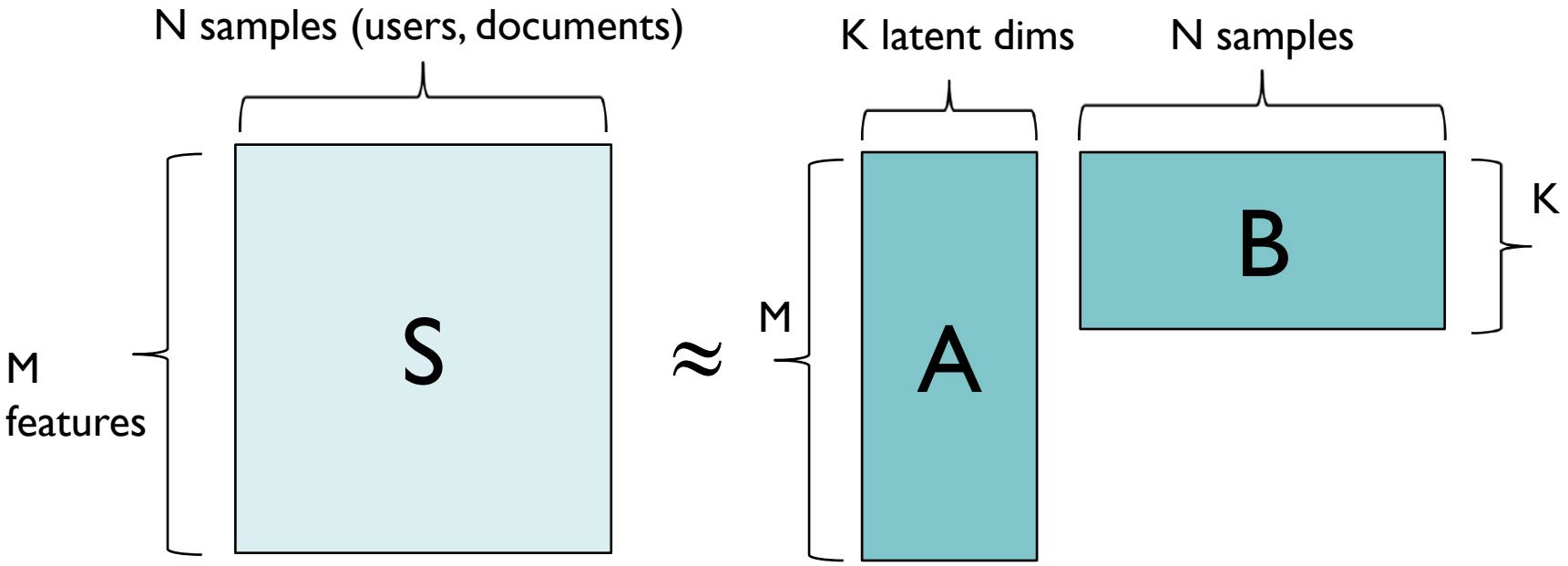
Motivation

Usually, data vary more strongly in some dimensions than others. Fitting a linear model to the (k) strongest directions gives the best approximation to the data in a least-squares sense.



Motivation

Algebraically, we have a high-dimensional data matrix (typically sparse) S, which we approximate as a product of two dense matrices A and B with low "inner" dimension:



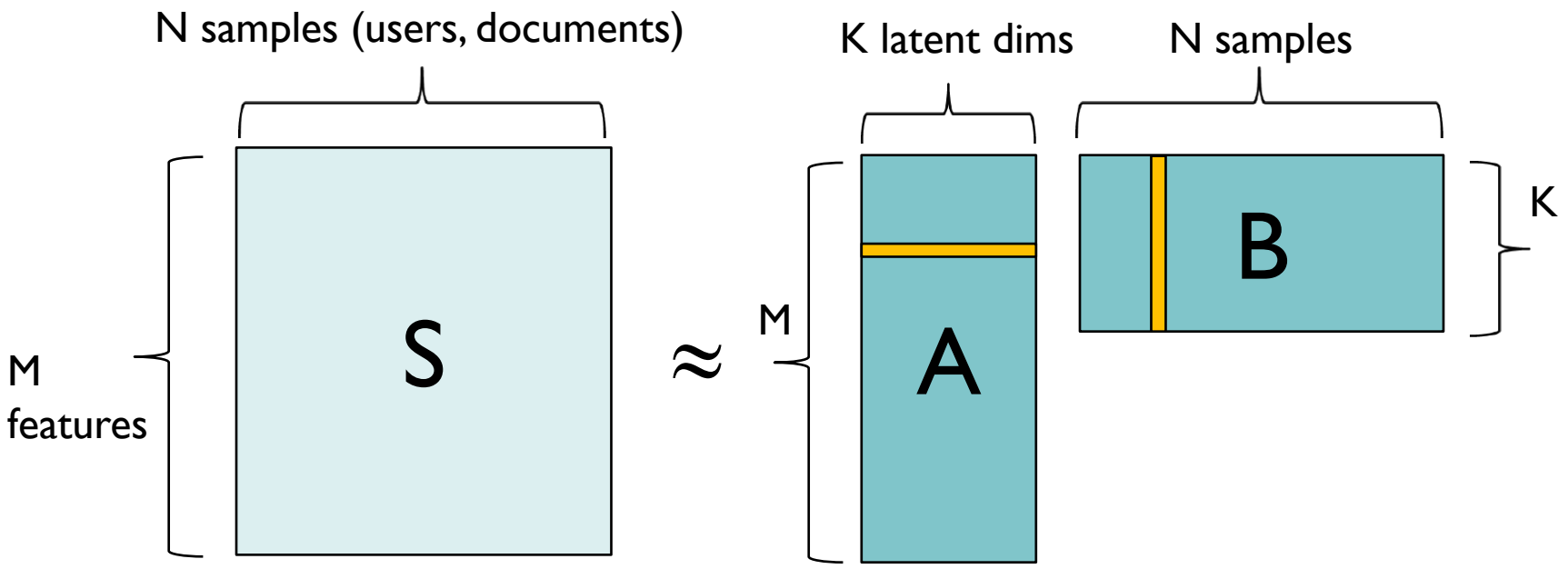
From the factorization, we can fill in missing values of S. This problem is often called "Matrix Completion".

Motivation

Columns of B represent the distribution of topics the n^{th} sample.

Rows of A represent the distribution of topics in the m^{th} feature.

These weights allow us to interpret the latent dimensions.



Outline

- Motivation
- Linear subspaces: SVD and PCA
- LSI
- Netflix/Alternating Least Squares
- Non-negative Matrix Factorization
- Topic Models: LDA

Eigenvalues

- Recall that the *eigenvalues* for a symmetric matrix A are values λ satisfying:

$$Ax = \lambda x$$

and the vectors *x* are the *eigenvectors*.

- If the eigenvalues are all positive, the matrix is positive definitive.

- A positive-definite matrix admits an eigendecomposition, which is a factorization as

$$A = UDU^T$$

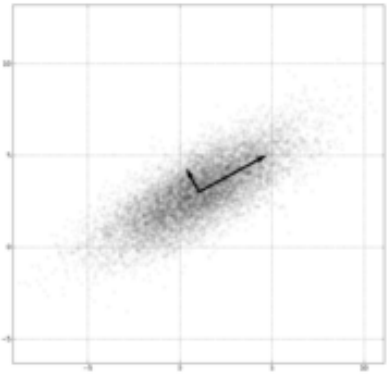
where *D* is a diagonal matrix of the eigenvalues and *U* is an orthonormal matrix whose columns are the eigenvectors.

Principal Component Analysis

- The principal components of a set of data can be found from the (first k) eigenvalues and eigenvectors of the covariance matrix, $\mathbf{Q} = [q_{jk}]$

$$q_{jk} = \frac{1}{N-1} \sum_{i=1}^{N} (x_{ij} - \bar{x}_j)(x_{ik} - \bar{x}_k)$$

- For a given k, the k principal components "explain" the maximum amount of variance in the data.
- They form a natural low-dimensional representation for the data.



Singular Value Decomposition

- PCA can be computed directly from the data matrix using Singular Value Decomposition (SVD). The SVD of an m x n matrix M is:

$$M = U\Sigma V^*$$

  Where *U* is an m x m unitary matrix, Σ is an m x n diagonal matrix of singular values and *V* is an n x n matrix unitary matrix.

- The columns of U are the left singular vectors and are the eigenvectors of MM^T
- The columns of V are the right singular vectors and are eigenvectors of M^TM
- The non-zero diagonal elements of are the square roots of the non-zero eigenvalues of MM^T or M^TM

PCA with SVD

- Given the singular value decomposition of *M*:

$$M = U\Sigma V^*$$

we can use the singular vectors to project from either space down to the subspace spanned by the first k singular values.

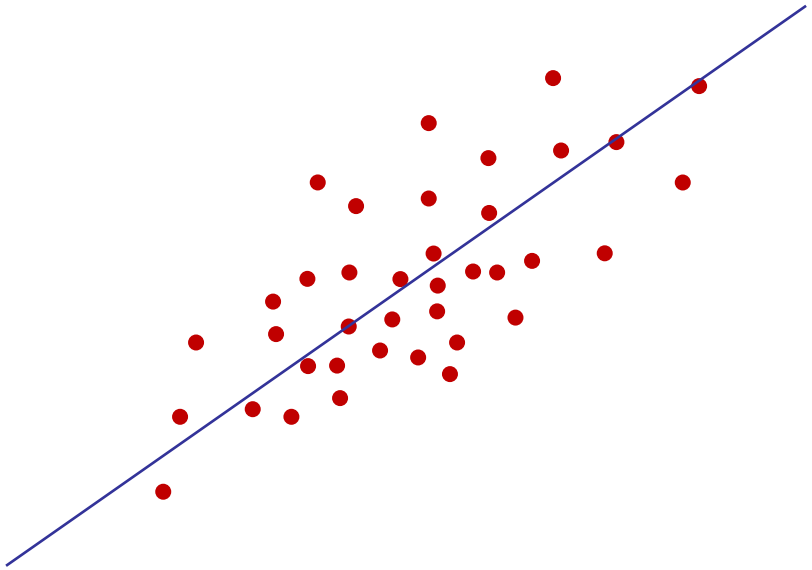
- e.g. m = number of terms, n = number of documents, and let U_k denote the matrix comprising the first k columns of *U*, V_k denote the matrix comprising the first k columns of *V*

- The projection of a document *X* is $X^T U_k$, while the projection of a term (given the vector *Y* of counts in all documents) is $V_k{}^T Y$

PCA incrementally

We could also construct PCA by:

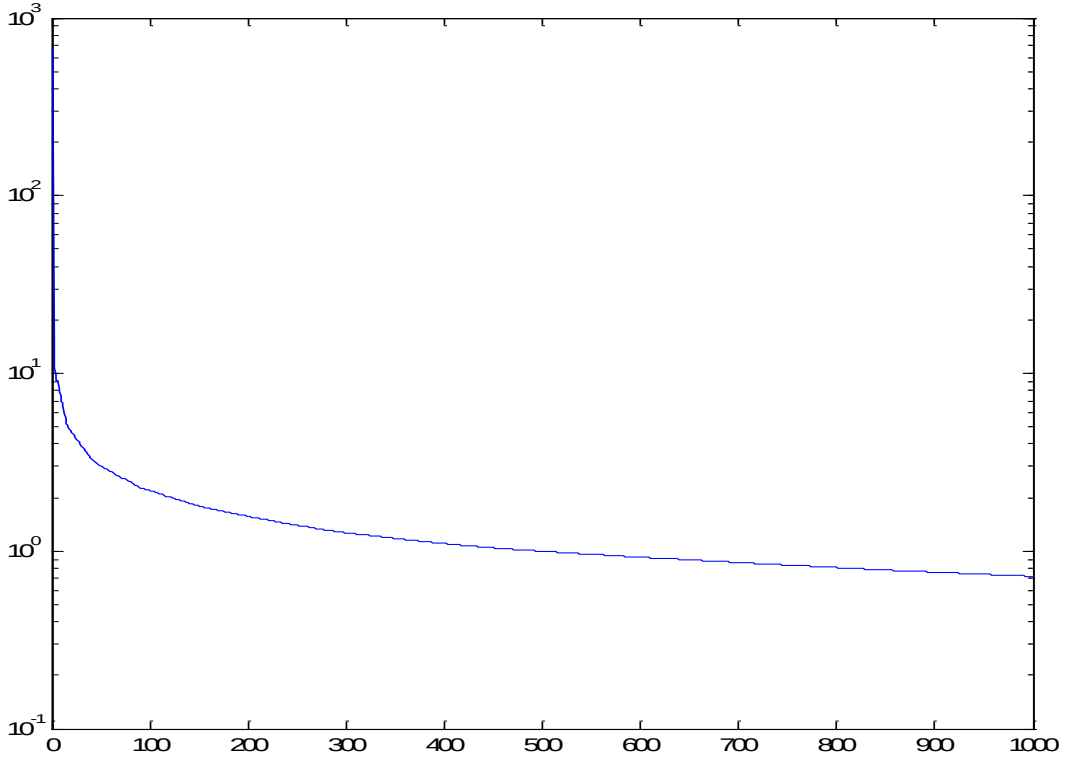
- Fitting a squared-distance minimizing line to the data
- Computing residuals normal to the line
- Iterating on the residuals



Example

Amazon review dataset, first 20 singular values:

662.7120
15.1315
10.8356
9.8964
9.1680
9.0699
8.2452
7.9291
7.5407
6.9114
6.8163
6.5669
5.9085
5.5684
5.1979
5.1280
4.9712
4.8355
4.7213
4.6493



Over 99% of the variance is explained by the first component

PCA uses

Given the low-dimensional representation from PCA, we can perform a variety of operations:

- Prediction
- Clustering
- Comparison (compute a similarity measure between two points)

The data are usually more compact (fewer coefficients) and dense, whereas the original data were sparse.

Using PCA with tf-idf

PCA projection depends on the metric properties of the space, and is affected by scaling.

i.e., the meaning of "orthogonal" changes with scaling of coordinates.

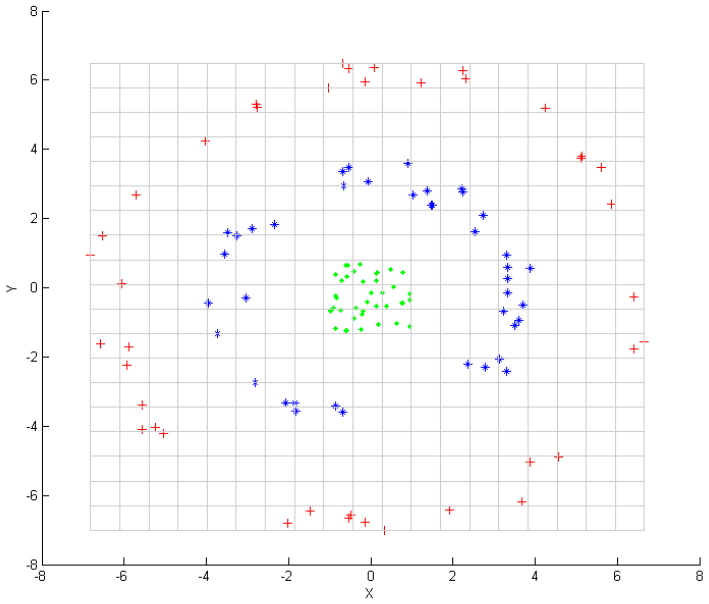
Scaling operations like tf-idf should be performed first.

tfidf = tf(t) * idf(t)

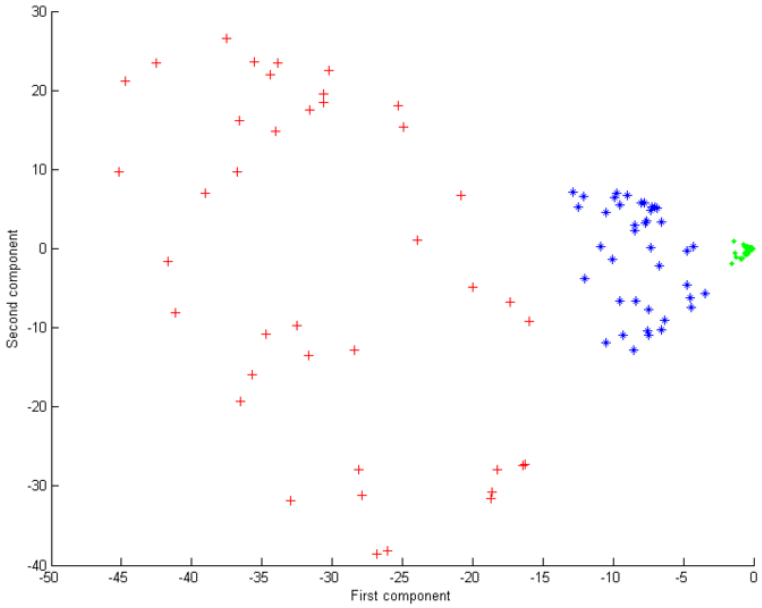
tf(t) = frequency of term t in a document, idf(t) = log(D/D(t)) where D is the total number of documents, and D(t) is the number of documents containing t.

kernel PCA

In fact PCA can be formulated to depend entirely on inner products (kernels), leading to kernel PCA.



Original point set



After kernel-PCA with quadratic kernel

Outline

- Motivation
- Linear subspaces: SVD and PCA
- LSI
- Netflix/Alternating Least Squares
- Non-negative Matrix Factorization
- Topic Models: LDA

LSI and PCA

- Latent Semantic Indexing is an approach to dimension reduction for text retrieval.
- First, term weights on documents are adjusted to tf-idf values.
- Then SVD/PCA is run on the corpus.
- Each document is projected down into the latent space, using the singular vector matrix.
- For retrieval, queries are projected into the latent space, and query-document matching is determined by a metric in that space, such as cosine distance.

LSI and PCA

- Results circa 1997

Test Collection	LSI	keyword	measure
Med-e	.66	.51	average precision
Med	.52	.46	average precision
Cran	.39	.29	average precision
ADI	.29	.26	average precision
Cisi	.11	.11	average precision
News	.61	.55	average precision
TM	.40	.35	average precision
TREC	.30	.26	average precision
TREC	8676	8043	number relevant
Toefl	53.5	29.5	number correct

LSI and PCA

- Compared to direct keyword matching, LSI improved both recall (usually) and sometimes precision.
- Both document and query descriptions are generally dense in the latent space, so matches are more frequent.
- The system can retrieve documents that don't contain a query term, assuming similar terms occur in the same PCA factor as the original word.

Outline

- Motivation
- Linear subspaces: SVD and PCA
- LSI
- Netflix/Alternating Least Squares
- Non-negative Matrix Factorization
- Topic Models: LDA

Alternating Least Squares

ALS (Alternating Least Squares) is a popular method for matrix factorization for sparse, linear data (ratings).

- Input matrix S is sparse, so we approximate it as a low-dimensional product: $S \approx A * B$

- Typically minimize quadratic loss on non-zeros of S

$$L_2(S - A *_S B)$$

and with L_2 regularizers on A and B. Here $*_S$ denotes the product evaluated only at non-zeros of S.

Alternating Least Squares

We can compute gradients with respect to A and B which are:

$$\frac{dl}{dA} = -2(S - A *_S B)B^T + 2w_A A$$

where w_A is the weighting of the regularizer on A, and

$$\frac{dl}{dB} = -2A^T(S - A *_S B) + 2w_B B$$

where w_B is the weight of the regularizer on B.

Then the loss can be minimized with SGD. However, this can be quite slow (high dimensional model, mutual dependence of A,B).

ALS closed form

Given A, solving for a column of B is a standard linear regression task, and has a closed form:

$$B_i = (w_B I - A^T D_i A)^{-1} A^T S_i$$

where S_i is the i^{th} column of S, D_i is a diagonal matrix with ones at the non-zeros rows of S_i.

A similar formula exists for rows of A.

Because of the matrix inversions, this method is quite expensive requiring $O(K^3(N+M))$ time. This is however the standard implementation (e.g. Mahout, Powergraph).

ALS semi-closed form

Instead, write the expression for B_i as a linear equation:

$$A^T S_i = (w_B I - A^T D_i A) B_i$$

Since B_i should change slightly on each ALS major iteration, we can instead use an ***iterative matrix solver*** (e.g. conjugate gradient). This cuts the complexity from $O(K^3(N+M))$ to $O(K^2(N+M))$. But we can actually do better.

The above formula, when written for all columns, becomes:

$$A^T S = w_B B - A^T (A *_S B) = lf(B)$$

Where $lf(B)$ is a linear function of B. We can use a "black-box" conjugate gradient solver for each column of B. This reduces the complexity down to O(KP) where P is the number of non-zeros in S. This is much faster for typical K (1000 for Netflix).

Netflix

- From the first year onwards all competitive entries used a dimension reduction model.
- The dimension reduction accounts for about half the algorithms' gain over the baseline.
- The winning team's model is roughly:

$$\hat{r}_{ui} = b_{ui} + q_i^T \left(|\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) x_j + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right)$$

- r is the rating, b is a baseline, q is an item factor, the remainder is the "user factor".
- It incorporates new factor vectors x: for explicit ratings and y: for implicit ratings.

Netflix Factor Model

- Factors are found iteratively, by direct gradient minimization.
- Typically ran for around 30 iterations.
  - $b_u \leftarrow b_u + \gamma_1 \cdot (e_{ui} - \lambda_6 \cdot b_u)$
  - $b_i \leftarrow b_i + \gamma_1 \cdot (e_{ui} - \lambda_6 \cdot b_i)$
  - $q_i \leftarrow q_i + \gamma_2 \cdot (e_{ui} \cdot (p_u + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j) - \lambda_7 \cdot q_i)$
  - $p_u \leftarrow p_u + \gamma_2 \cdot (e_{ui} \cdot q_i - \lambda_7 \cdot p_u)$
  - $\forall j \in \mathrm{N}(u)$:
    $y_j \leftarrow y_j + \gamma_2 \cdot (e_{ui} \cdot |\mathrm{N}(u)|^{-\frac{1}{2}} \cdot q_i - \lambda_7 \cdot y_j)$
  - $\forall j \in \mathrm{R}^k(i;u)$:
    $w_{ij} \leftarrow w_{ij} + \gamma_3 \cdot \left(|\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \cdot e_{ui} \cdot (r_{uj} - b_{uj}) - \lambda_8 \cdot w_{ij} \right)$
  - $\forall j \in \mathrm{N}^k(i;u)$:
    $c_{ij} \leftarrow c_{ij} + \gamma_3 \cdot \left(|\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \cdot e_{ui} - \lambda_8 \cdot c_{ij} \right)$

Netflix

- Summary of dimension reduction models:

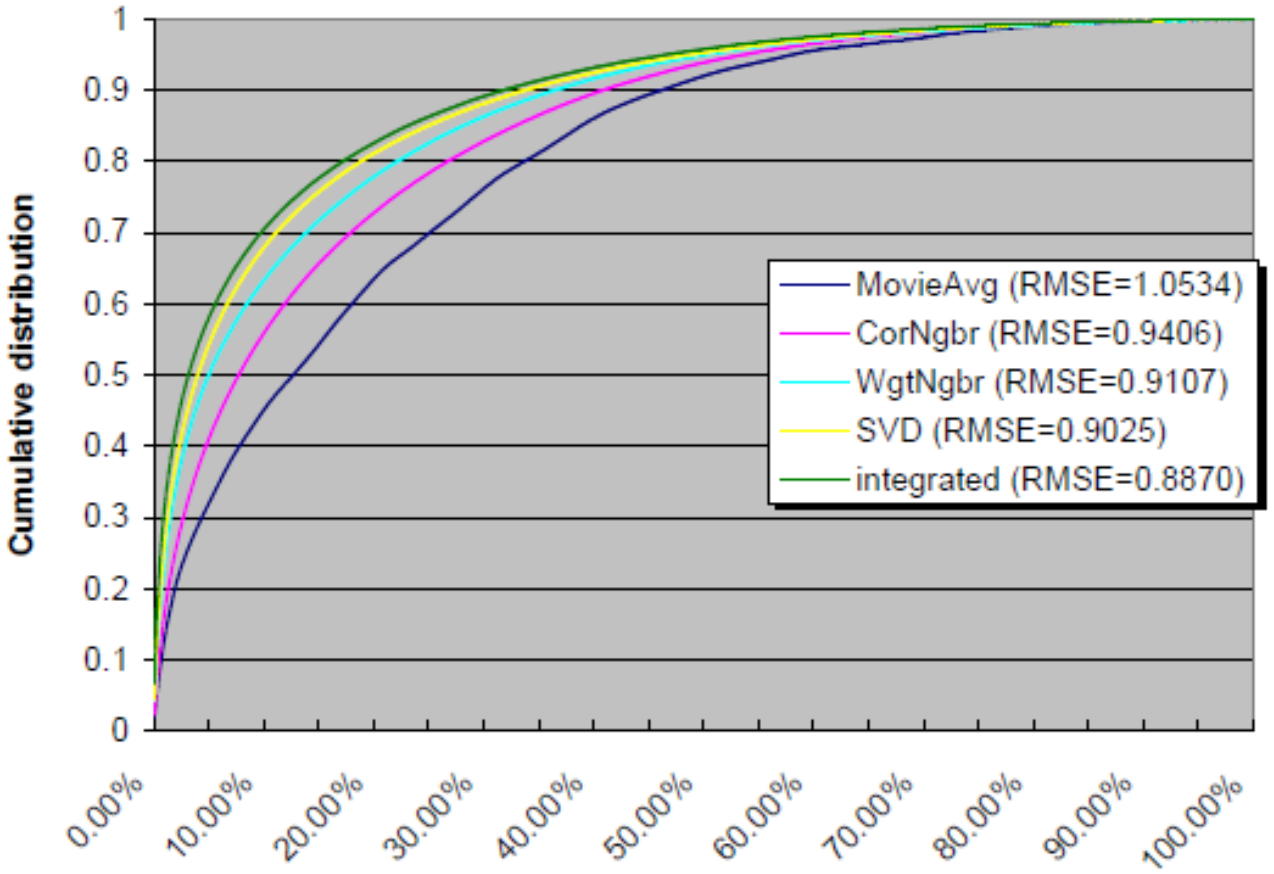
Model	50 factors	100 factors	200 factors
SVD	0.9046	0.9025	0.9009
Asymmetric-SVD	0.9037	0.9013	0.9000
SVD++	0.8952	0.8924	0.8911

- Dimension reduction augmented by neighborhoods:

	50 factors	100 factors	200 factors
RMSE	0.8877	0.8870	0.8868
time/iteration	17min	20min	25min

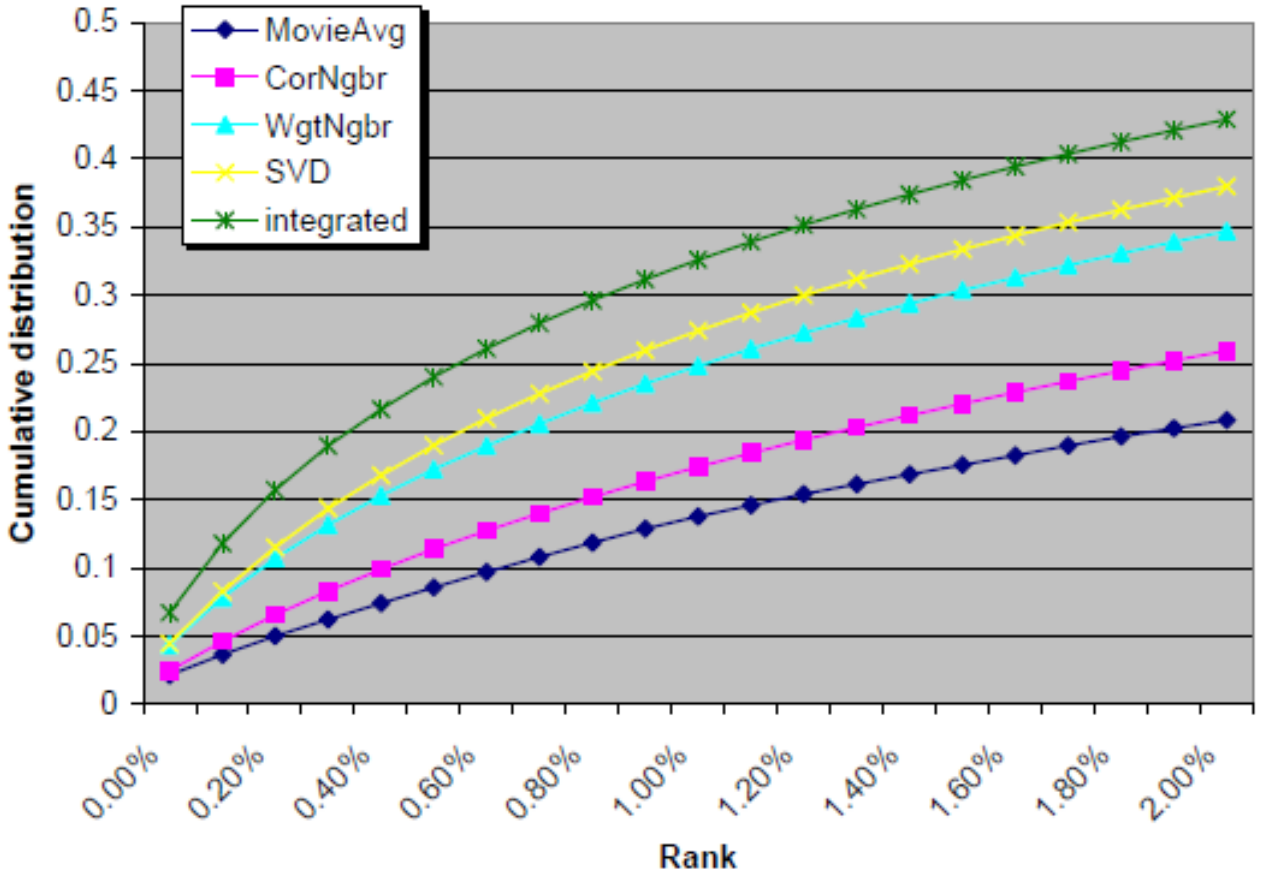
Netflix

- On top-k recommendations:



Netflix

- On top-k recommendations:



Outline

- Motivation
- Linear subspaces: SVD and PCA
- LSI
- Netflix/Alternating Least Squares
- Non-negative Matrix Factorization
- Topic Models: LDA

Non-Negative Matrix Factorization

There is an input matrix S is sparse that we want to approximate with a low-dimensional factorization $S \approx A * B$

- The key constraint is that ***coefficients of A and B are non-negative***.

- The loss function may be L_2 loss, or it may be a "likelihood" loss or generalized KL-divergence.

NMF models cases where there sample-topic and feature-topic weights are naturally non-negative (e.g. texts, computer vision).

Non-Negative Matrix Factorization

NMF models cases where there sample-topic and feature-topic weights are naturally non-negative (e.g. texts, computer vision).

Data matrices are often count data (words or feature counts).

Sparseness in the data matrices implies zero counts (absence of rare terms), rather than missing or unknown values.

NMF Loss Functions

L2 loss:

$$l = \sum_{i,j} \left(S_{i,j} - (AB)_{i,j}\right)^2$$

Generalized KL-divergence:

$$l = \sum_{i,j} \left(S_{i,j} \log \frac{S_{i,j}}{(AB)_{i,j}} - S_{i,j} + (AB)_{i,j} \right)$$

KL-divergence is a measure of dissimilarity between S and AB treated as discrete probability distributions.

NMF Solution

Finding a minimum-loss model is more difficult with non-negative coefficients: straightforward gradient implementations will violate the constraints.

While there are no direct methods to find NMF factorizations for given dimension k, luckily there are ***alternating multiplicative update*** formulas for A and B.

These formulae converge quite fast in practice (few iterations), but entail multiple full passes over the dataset.

NMF update formulae

L2 loss:

$$A \leftarrow A \circ \frac{SB^T}{ABB^T}$$

$$B \leftarrow B \circ \frac{A^T S}{A^T AB}$$

KL-divergence loss:

$$A \leftarrow A \circ \left(\frac{S}{AB} B^T\right) / \mathrm{sum}(A, 1)$$

$$B \leftarrow B \circ \left(A^T \frac{S}{AB}\right) / \mathrm{sum}(B, 2)$$

NMF update formulae

Complexity is $O(K^2(N+M))$ for L2-NMF

For KL-divergence NMF, we can simplify the calculation to:

$$A \leftarrow A \circ \left(\frac{S}{A *_S B} B^T \right) / \mathrm{sum}(A, 1)$$

which requires only O(KP) time, where P = number of non-zeros of S.

Outline

- Motivation
- Linear subspaces: SVD and PCA
- LSI
- Netflix/Alternating Least Squares
- Non-negative Matrix Factorization
- Topic Models: LDA

Latent Dirichlet Allocation

There are k topics

A document of N words $\boldsymbol{w} = \langle w_1, \ldots, w_N \rangle$ is generated as:

- Choose θ from Dirichlet($\alpha_1, \ldots, \alpha_k$)
- For each word:
  - Choose a topic z_n from Mult(θ)
  - Choose the word w_n from $p(w|z_n)$

- The document probability is:

$$p(\mathbf{w}) = \int_\theta \left(\prod_{n=1}^{N} \sum_{z_n=1}^{k} p(w_n|z_n;\beta) p(z_n|\theta) \right) p(\theta;\alpha) d\theta$$

Latent Dirichlet Allocation

There are k topics

A document of N words $\boldsymbol{w} = \langle w_1, \ldots, w_N \rangle$ is generated as:

- Choose θ from Dirichlet($\alpha_1, \ldots, \alpha_k$)
- For each word:
  - Choose a topic z_n from Mult(θ)
  - Choose the word w_n from $p(w|z_n)$

- The document probability is:

$$p(\mathbf{w}) = \int_\theta \left(\prod_{n=1}^{N} \sum_{z_n=1}^{k} p(w_n|z_n;\beta) p(z_n|\theta) \right) p(\theta;\alpha) d\theta$$

A B

Latent Dirichlet Allocation

Recurrences:

$$\begin{aligned}
\phi_{ni} &\propto \beta_{iw_n} \exp\left(\Psi(\gamma_i) - \Psi\left(\sum_{j=1}^{k} \gamma_j\right)\right) \\
\gamma_i &= \alpha_i + \sum_{n=1}^{N} \phi_{ni} \\
\beta_{ij} &\propto \sum_{m=1}^{M} \sum_{n=1}^{|\mathbf{w}_m|} \phi_{mni} w_{mn}^{j}
\end{aligned}$$

produce the word/topic model β and approximations γ to the document/topic means. In practice, collapse identical words to bag-of-words.

Complexity O(PKt) where M = total number of non-zero terms in all docs, k = number of dimensions, t = number of iterations.

LDA by Gibbs Sampling

Add a prior on the document/topic model:

$$
\begin{aligned}
w_i|z_i,\ \phi^{(z_i)} &\sim \text{Discrete}(\phi^{(z_i)}) \\
\phi &\sim \text{Dirichlet}(\beta) \\
z_i|\theta^{(d_i)} &\sim \text{Discrete}(\theta^{(d_i)}) \\
\theta &\sim \text{Dirichlet}(\alpha)
\end{aligned}
$$

Then its possible to explore the parameter space (φ, θ) using Monte-Carlo methods (a Gibbs sampler). Each word assignment is updated according to:

$$
P(z_i = j|\mathbf{z}_{-i}, \mathbf{w}) \propto \frac{n_{-i,j}^{(w_i)} + \beta}{n_{-i,j}^{(\cdot)} + W\beta} \frac{n_{-i,j}^{(d_i)} + \alpha}{n_{-i,\cdot}^{(d_i)} + T\alpha}
$$

LDA by Gibbs Sampling

After running the sampler for a number of steps, the parameters are updated as:

$$\hat{\phi}_j^{(w)} = \frac{n_j^{(w)} + \beta}{n_j^{(\cdot)} + W\beta}$$

$$\hat{\theta}_j^{(d)} = \frac{n_j^{(d)} + \alpha}{n_\cdot^{(d)} + T\alpha}.$$

Since only the non-zero counts need to be stored, the sampler uses less space than full (variational) inference.

Multiple chains taken. Warm-up time, "lag" and then actual samples are taken.

Mixture Models vs. Clustering

LDA using variational inference computes complete (dense) parameters β and γ, so this is a true mixture model. This limits the scale of models for typical document or user data to 1k dims or so.

The Gibbs sampler implementation for LDA can scale further (number of non-zeros roughly = number of steps), so it behaves more like a soft clustering algorithm. Communication is a problem for the standard implementation though.

Summary

- Linear subspaces: SVD and PCA
- LSI
- Netflix/Alternating Least Squares
- Non-negative Matrix Factorization
- Topic Models: LDA