Week 11

Agenda

1. Eigenfaces paper discussion
2. PCA discussion
3. PCA notebook

For next week: Read Barabasi paper (or textbook: http://networksciencebook.com/)

Representation Discussion

1. What is 'representation' in machine learning?
2. What is the representation used in the Iris dataset?
3. What is the representation used in the MNIST dataset?
4. What was the representation we used for spam classification?
5. What are some 'good' qualities of a representation?
6. What is feature selection? What are some ways to do it? Why do it?
7. Are the nodes on the hidden layers of a neural network features? What happens if the hidden layer is smaller than the input layer?
8. What does PCA do to a representation?

Goal

- Reduce feature space from 65536 pixels to 7 'eigenface' weights

Pre-processing

- Compute the mean for each pixel and do mean normalization of the training data.

Note

- Faces must be in same position with same background for this to work (as opposed to Convolutional Nets).

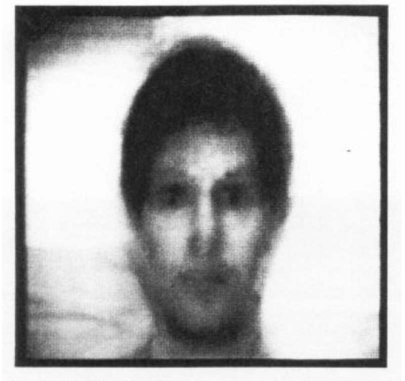


Figure 1. **(b)** The average face Ψ.

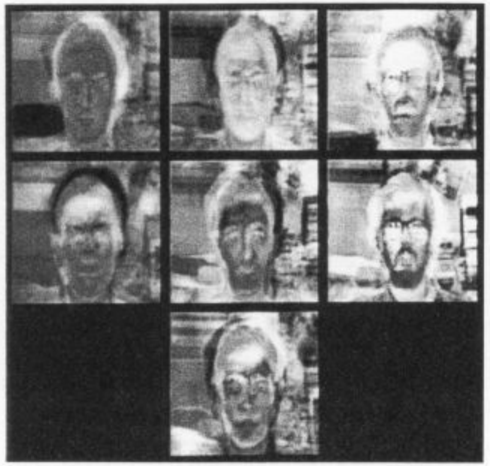


Figure 2. Seven of the eigenfaces calculated from the input images of Figure 1.

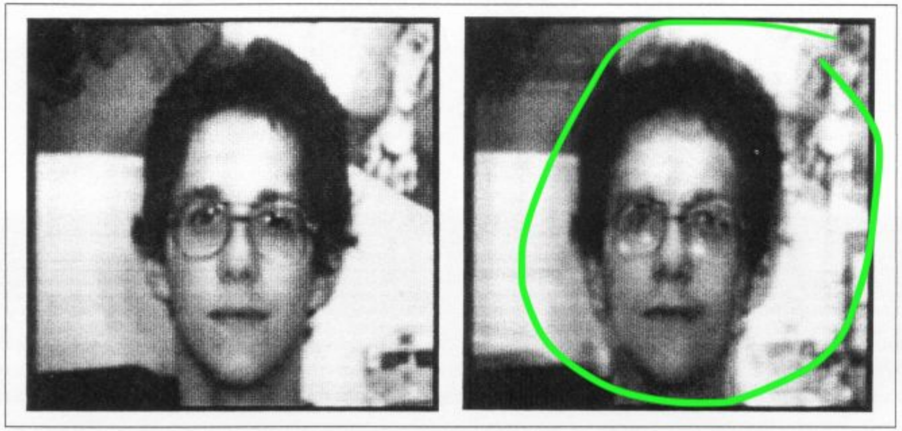


Figure 3. An original face image and its projection onto the face space defined by the eigenfaces of Figure 2.

Covariance Matrix

- Similar to GMM, PCA is an operation on the covariance matrix of the training data
- There is an improved version which allows for using a matrix smaller than the full covariance matrix (65536*65536)

SVD

- Compute the SVD using one of the methods (e.g. Power iteration)
- The result (U) provides a set of eigenvectors ordered by their corresponding eigenvalues.
- The seven eigenvectors with the largest eigenvalues are shown. These are reffered to as 'principle components'

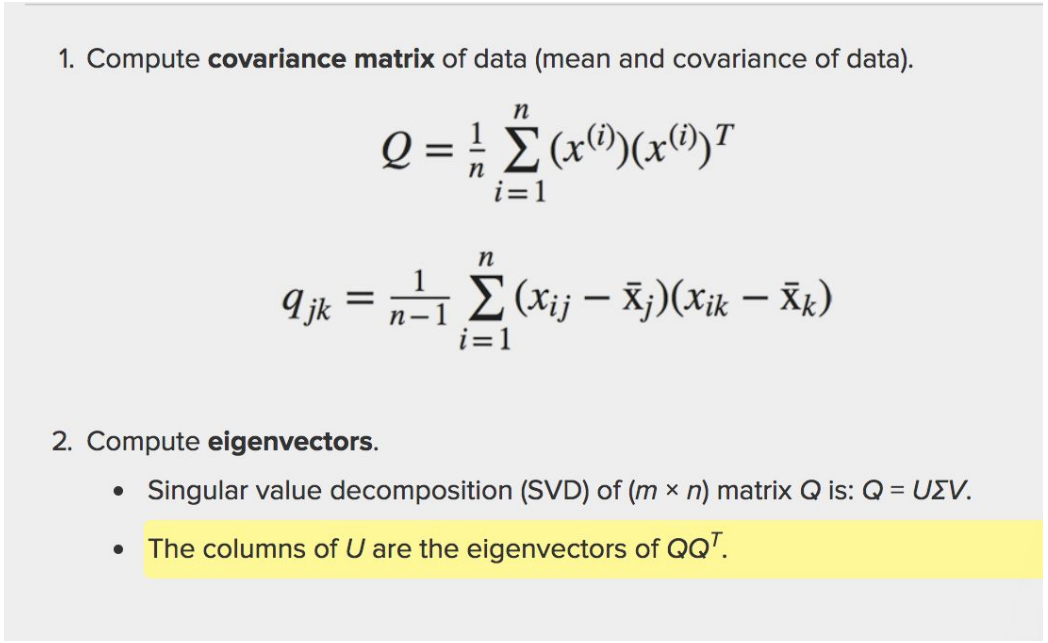
Steps to reduce dimensionality of new images

- Subtract the training mean from each pixel
- For each of the seven eignenvectors, multiply the image vector by the eigenvector to get a wieght
- Each new image is thus represented simply as 7 weights (loadings)

Dim Reduction

- Store the training images as weight vectors. Train a classifier (e.g. SVM) with those wieghts as features.
- http://scikit-learn.org/stable/auto_examples/applications/face_recognition.html

PCA



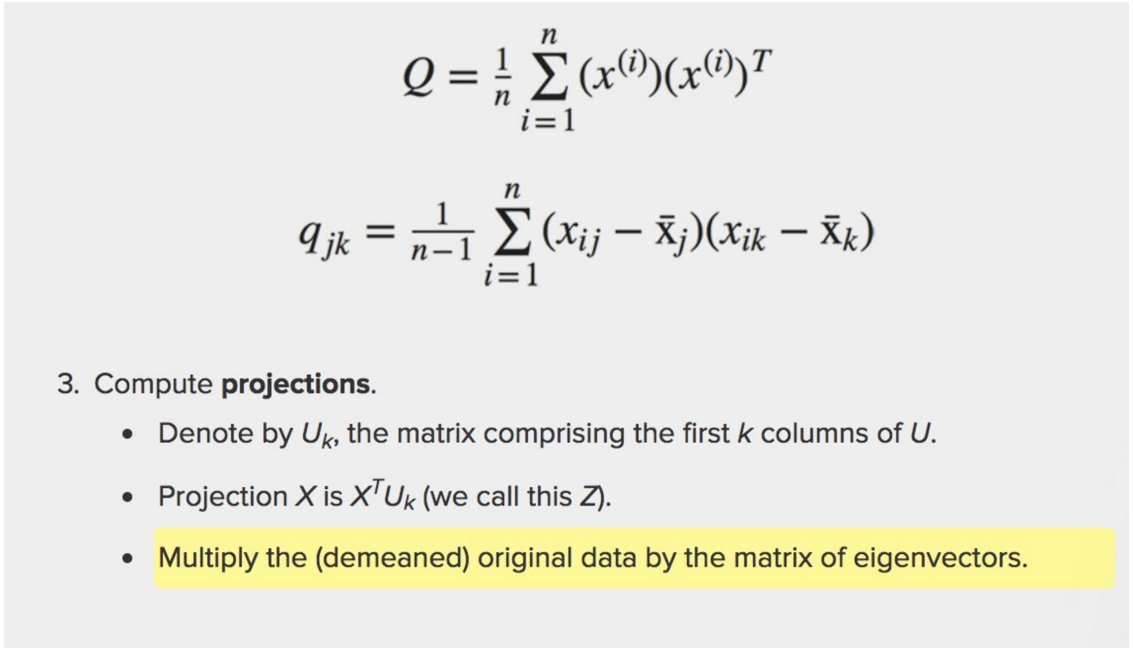
1. Compute **covariance matrix** of data (mean and covariance of data).

$$Q = \frac{1}{n} \sum_{i=1}^{n} (x^{(i)})(x^{(i)})^T$$

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

2. Compute **eigenvectors**.
   - Singular value decomposition (SVD) of ($m \times n$) matrix Q is: $Q = U\Sigma V$.
   - The columns of U are the eigenvectors of QQ^T.

PCA

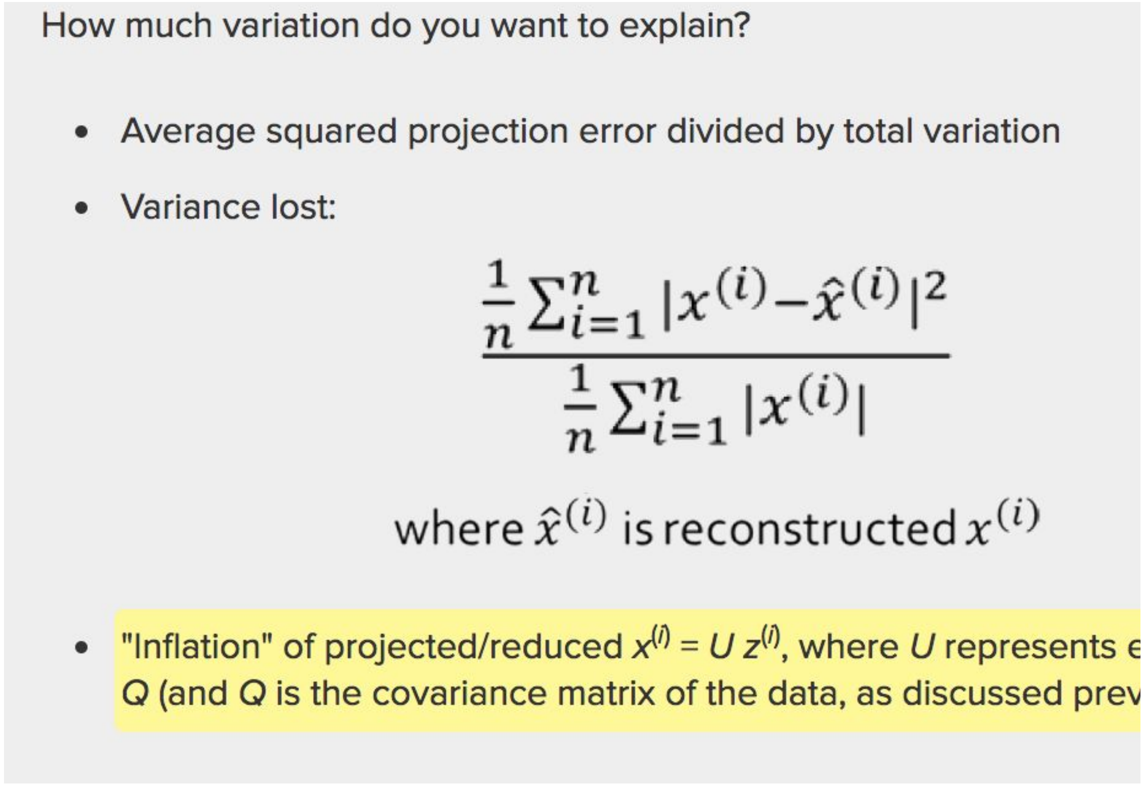


$$Q = \frac{1}{n} \sum_{i=1}^{n} (x^{(i)})(x^{(i)})^T$$

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

3. Compute **projections**.
   - Denote by U_k, the matrix comprising the first k columns of U.
   - Projection X is $X^T U_k$ (we call this Z).
   - Multiply the (demeaned) original data by the matrix of eigenvectors.

PCA



How much variation do you want to explain?

- Average squared projection error divided by total variation
- Variance lost:

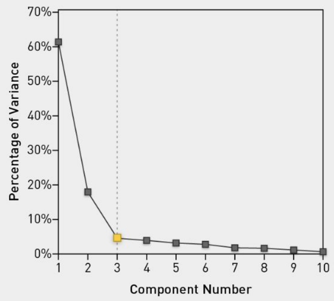
$$\frac{\frac{1}{n}\sum_{i=1}^{n}|x^{(i)}-\hat{x}^{(i)}|^2}{\frac{1}{n}\sum_{i=1}^{n}|x^{(i)}|}$$

where $\hat{x}^{(i)}$ is reconstructed $x^{(i)}$

- "Inflation" of projected/reduced $x^{(i)} = U z^{(i)}$, where U represents e
  Q (and Q is the covariance matrix of the data, as discussed prev

Example: Components of a 10-Dimensional Space

Axis	Variance	Cumulative
1	61.2%	61.2%
2	18.0%	79.2%
3	4.7%	83.9%
4	4.0%	87.9%
5	3.2%	91.1%
6	2.9%	94.0%
7	2.0%	96.0%
8	1.7%	97.7%
9	1.4%	99.1%
10	0.9%	100.0%



- Each component is an axis (table indicates share of variance accounted for).
- Three PCs account for roughly 84% of variance of original data.

Scaling SVD

HOW-TO

- Many ways! Choice depends on size of data, sparisity, etc.
- Most machine learning textbooks punt on the topic.
- For unrealistically small matrices, solve a system of equations by hand:
- http://web.mit.edu/be.400/www/SVD/Singular_Value_Decomposition.htm

POWER METHODS for SVD

- O(D^3)
- Compute one eigenvector at a time
- https://en.wikipedia.org/wiki/Power_iteration
- https://en.wikipedia.org/wiki/Lanczos_algorithm

SGD METHODS

- Popular for collaborative filtering as D is high
- Approximate the SVD
- Famous example: http://sifter.org/~simon/journal/20061211.html

RECENT METHODS

- Randomized Matrix Approximation
- https://research.facebook.com/blog/294071574113354/fast-randomized-svd/

1. U is an $m \times r$ *column-orthonormal matrix*; that is, each of its columns is a unit vector and the dot product of any two columns is 0.
2. V is an $n \times r$ column-orthonormal matrix. Note that we always use V in its transposed form, so it is the rows of V^T that are orthonormal.
3. Σ is a diagonal matrix; that is, all elements not on the main diagonal are 0. The elements of Σ are called the *singular values* of M.

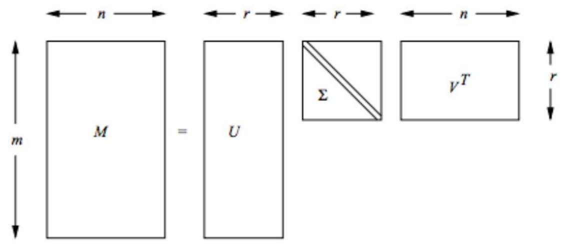
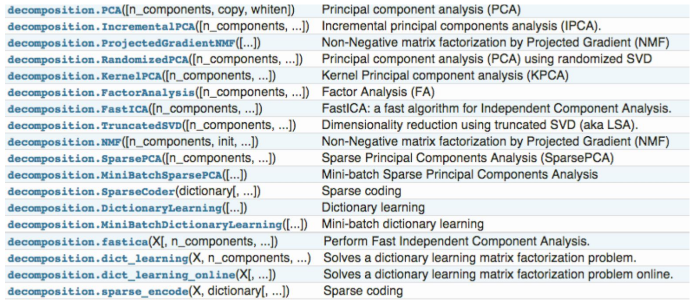
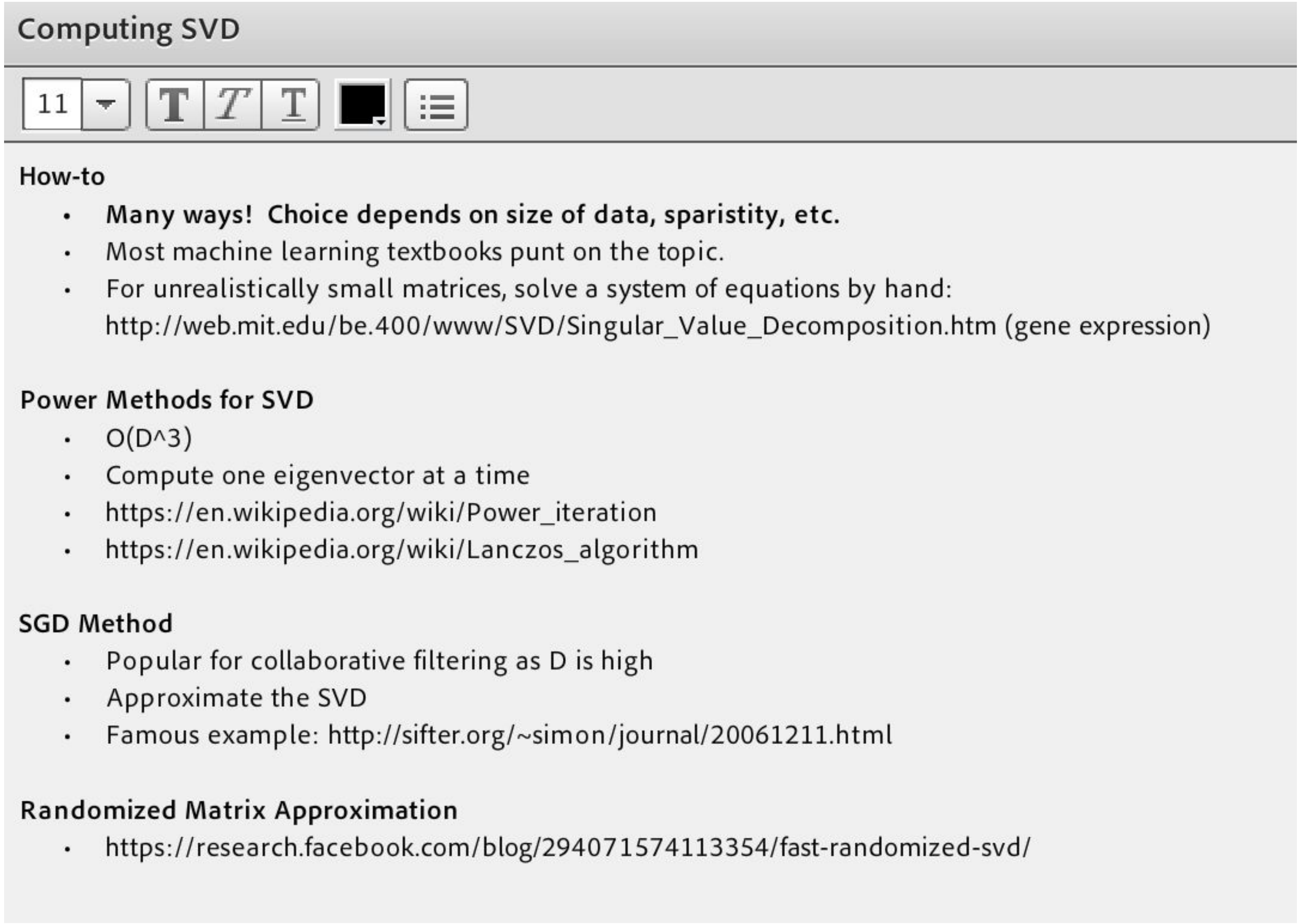


Figure 11.5: The form of a singular-value decomposition



decomposition.PCA([n_components, copy, whiten])	Principal component analysis (PCA)
decomposition.IncrementalPCA([n_components, ...])	Incremental principal components analysis (IPCA).
decomposition.ProjectedGradientNMF([...])	Non-Negative matrix factorization by Projected Gradient (NMF)
decomposition.RandomizedPCA([n_components, ...])	Principal component analysis (PCA) using randomized SVD
decomposition.KernelPCA([n_components, ...])	Kernel Principal component analysis (KPCA)
decomposition.FactorAnalysis([n_components, ...])	Factor Analysis (FA)
decomposition.FastICA([n_components, ...])	FastICA: a fast algorithm for Independent Component Analysis.
decomposition.TruncatedSVD([n_components, ...])	Dimensionality reduction using truncated SVD (aka LSA).
decomposition.NMF([n_components, init, ...])	Non-Negative matrix factorization by Projected Gradient (NMF)
decomposition.SparsePCA([n_components, ...])	Sparse Principal Components Analysis (SparsePCA)
decomposition.MiniBatchSparsePCA([...])	Mini-batch Sparse Principal Components Analysis
decomposition.SparseCoder(dictionary[, ...])	Sparse coding
decomposition.DictionaryLearning([...])	Dictionary learning
decomposition.MiniBatchDictionaryLearning([...])	Mini-batch dictionary learning
decomposition.fastica(X[, n_components, ...])	Perform Fast Independent Component Analysis.
decomposition.dict_learning(X, n_components, ...)	Solves a dictionary learning matrix factorization problem.
decomposition.dict_learning_online(X[, ...])	Solves a dictionary learning matrix factorization problem online.
decomposition.sparse_encode(X, dictionary[, ...])	Sparse coding



Computing SVD

How-to

- **Many ways! Choice depends on size of data, sparistity, etc.**
- Most machine learning textbooks punt on the topic.
- For unrealistically small matrices, solve a system of equations by hand: http://web.mit.edu/be.400/www/SVD/Singular_Value_Decomposition.htm (gene expression)

Power Methods for SVD

- O(D^3)
- Compute one eigenvector at a time
- https://en.wikipedia.org/wiki/Power_iteration
- https://en.wikipedia.org/wiki/Lanczos_algorithm

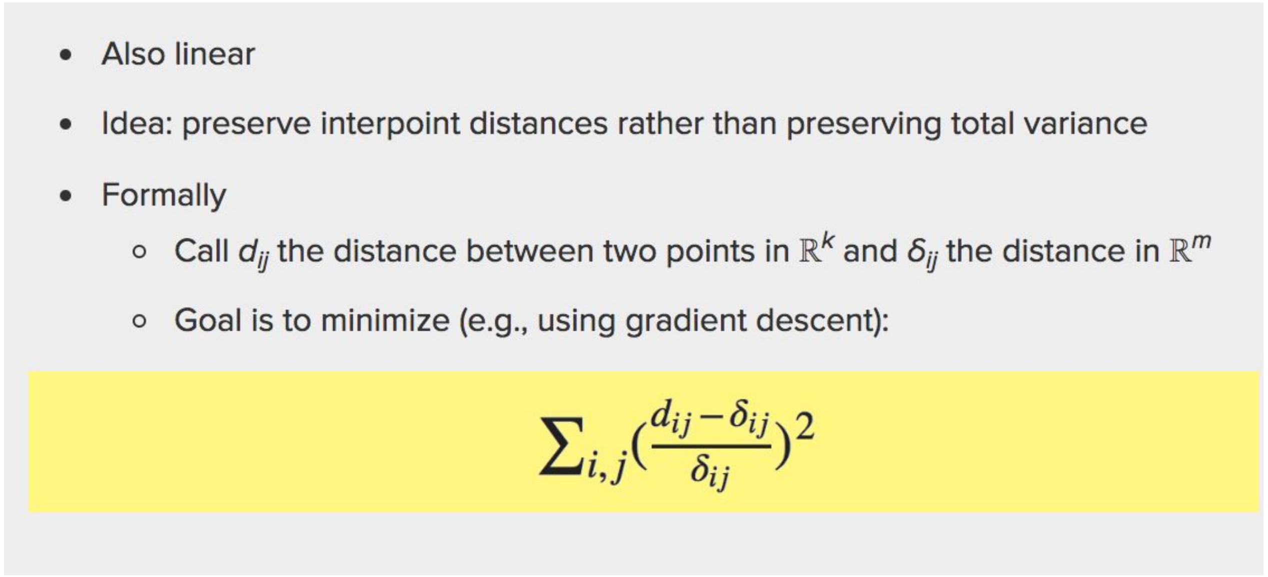
SGD Method

- Popular for collaborative filtering as D is high
- Approximate the SVD
- Famous example: http://sifter.org/~simon/journal/20061211.html

Randomized Matrix Approximation

- https://research.facebook.com/blog/294071574113354/fast-randomized-svd/

MDS



- Also linear
- Idea: preserve interpoint distances rather than preserving total variance
- Formally
  - Call d_{ij} the distance between two points in $\mathbb{R}^k$ and δ_{ij} the distance in $\mathbb{R}^m$
  - Goal is to minimize (e.g., using gradient descent):

$$\sum_{i,j}\left(\frac{d_{ij}-\delta_{ij}}{\delta_{ij}}\right)^2$$

Final Thoughts?