Dimensionality Reduction

COMP 527 Data Mining

Danushka Bollegala

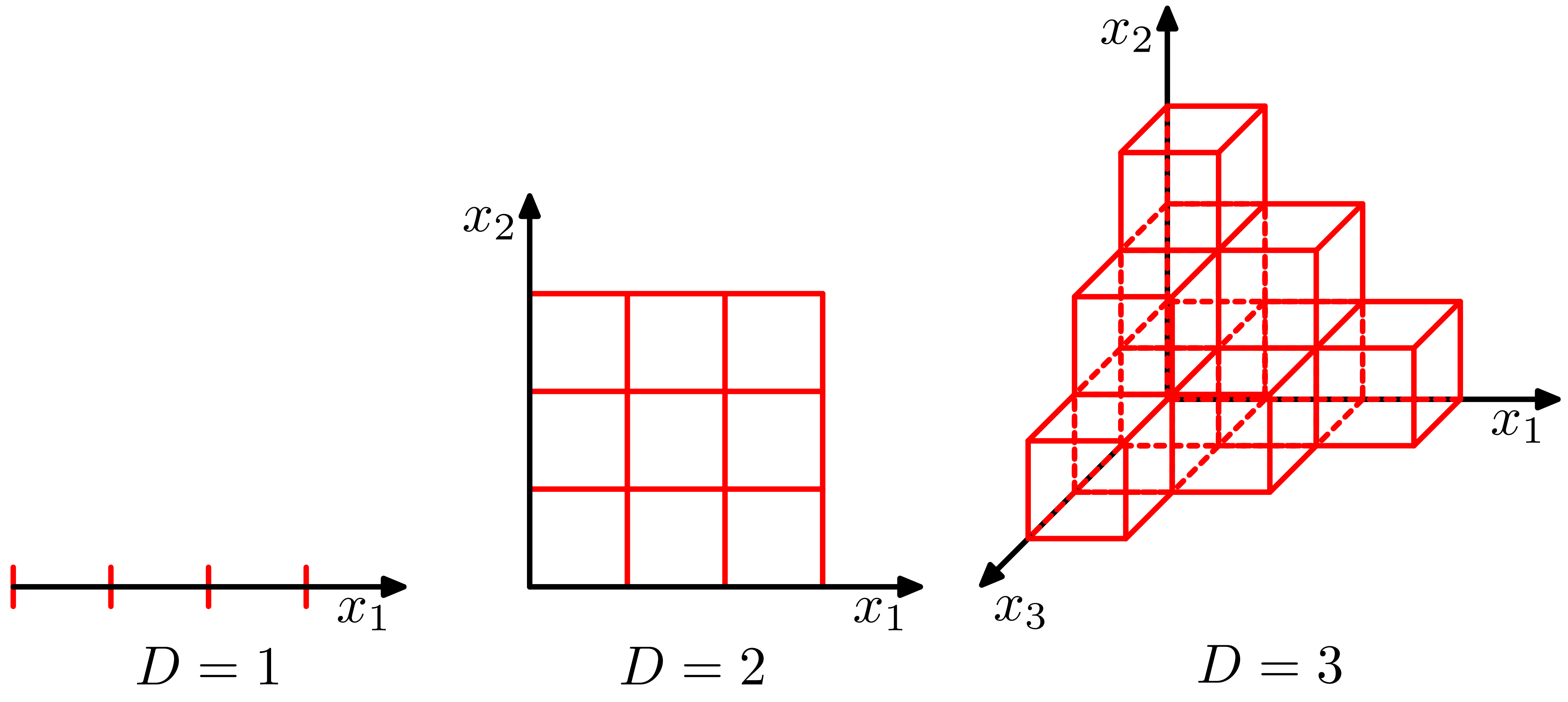
UNIVERSITY OF LIVERPOOL

Outline

- Problems with high dimensional data
- Dimensionality Reduction Methods
    - Singular Value Decomposition (SVD)
    - Principal Component Analysis (PCA)

Problems in High Dimensions

- Curse of dimensionality
  - We need exponentially large number of data points to cover a high dimensional space

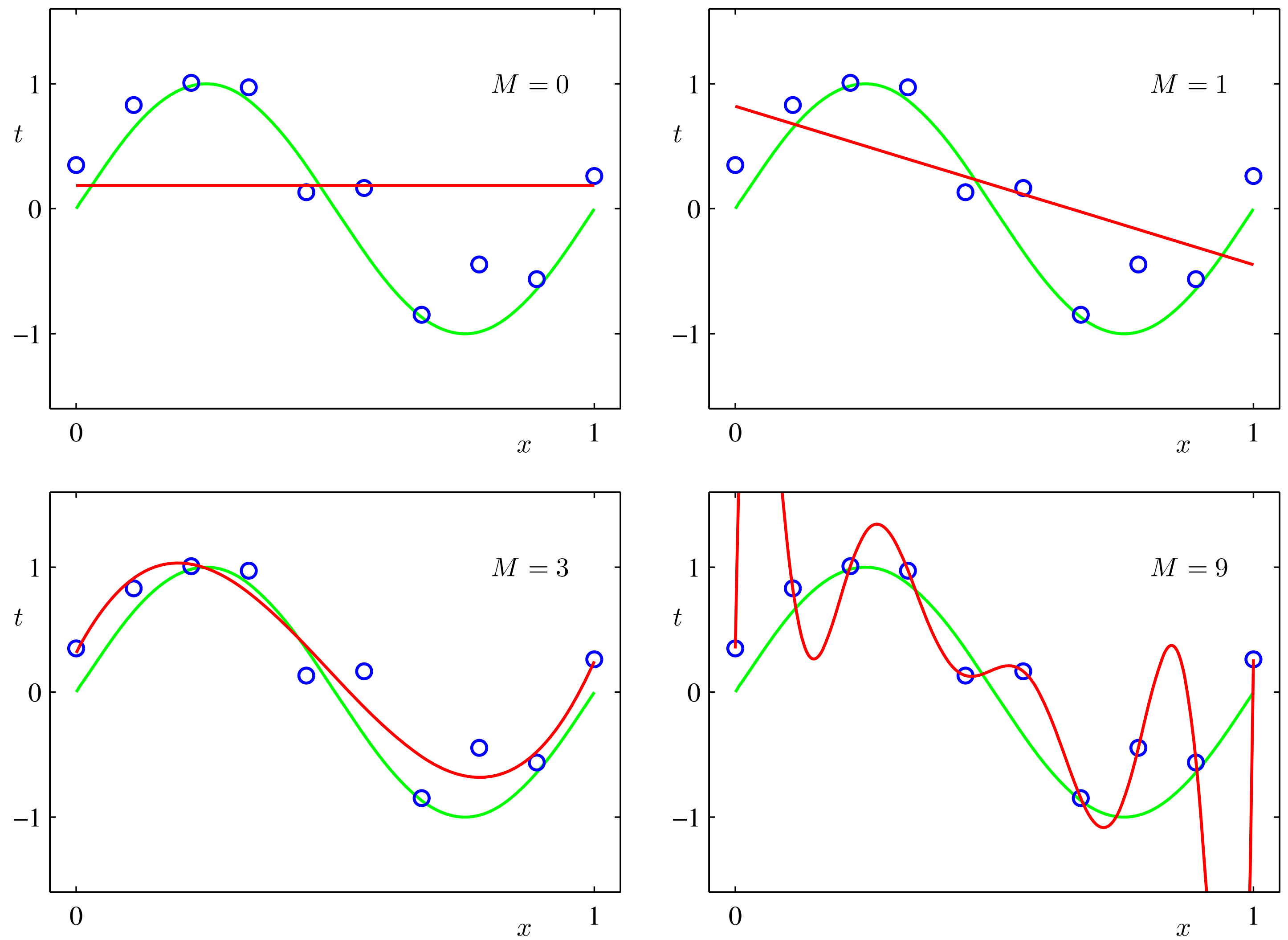


Problems in High Dimensions

- Data Sparseness
  - Although we have a large feature space (lots of dimensions to the data), we only observe a small number of non-zero features in any instance
  - This was the case for texts (in particular with the bag-of-words model)

Problems in High Dimensions

- Overfitting
  - Given n train data points, we can come up with an n-dimensional (n-th order) polynomial that passes through all those data points.
  - But it is very unlikely that it will fit well for the test data points



Problems in High Dimensions

- Time consuming (time complexity is large)
    - Consider computing cosine similarity between two n dimensional vectors, when n increases.
- Memory issues (space complexity is large)
    - Storing high dimensional dense vectors can be problematic when
        - the dimensionality of the vectors is large
        - there are lots of vectors (instances) to store

Solution

- Dimensionality Reduction
  - Try to *project* the original vectors to a *lower dimensional* space L
- What constraints do we have
  - Try to minimise the error due to the projection
    - If X and Y are neighbours in the original space, then they must also be neighbours in the projected space
  - Try to retain salient/important/principal dimensions as much as possible and remove the non-salient/ unimportant/auxiliary dimensions as much as possible

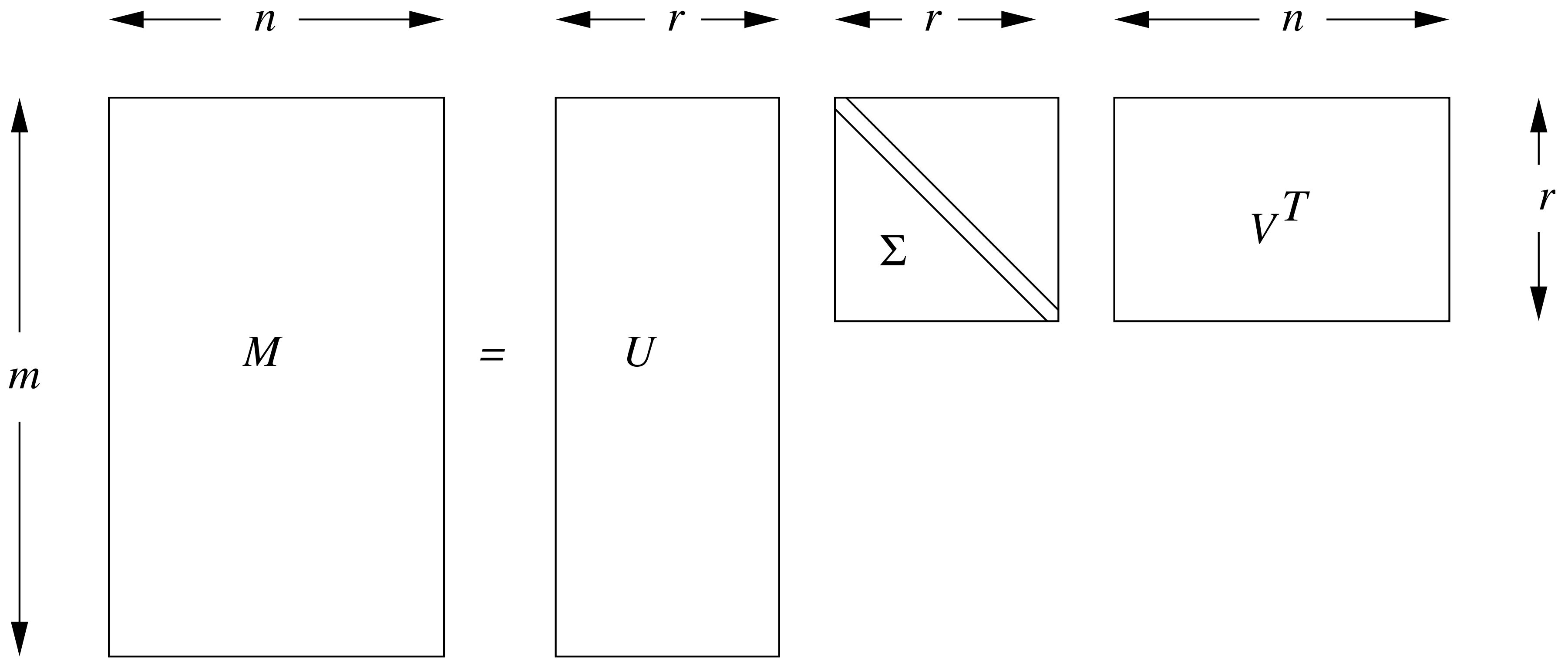
Eigenvalue Decomposition

- Linear Algebra Revision
  - Eigenvalues and Eigenvectors of a Square Matrix
  - A**x** $= \lambda$**x**
    - **x** is the eigenvector of A corresponding to the eigenvalue λ
- Compute the eigenvalues and eigenvectors of the following matrix

$$\begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix}$$

Singular Value Decomposition

- Eigenvalue decomposition can be performed only for square matrices.
- Singular Value Decomposition (SVD) is a operation that can be applied to any matrix
  - $M = U\Sigma V^T$
  - U and V are unitary (perpendicular) matrices, Σ is a diagonal matrix (singular values of M are diagonal elements of Σ).
  - Columns of U and V are perpendicular. $U^TU = I$ and $V^TV = I$.



SVD?

Snayperskaya Vintovka sistem'y Dragunova obraz’tsa (Dragunov Sniper Rifle or SVD)

Example

$$\underbrace{\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 0 & 0 & 2 & 2 \end{bmatrix}}_{M} = \underbrace{\begin{bmatrix} .14 & 0 \\ .42 & 0 \\ .56 & 0 \\ .70 & 0 \\ 0 & .60 \\ 0 & .75 \\ 0 & .30 \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} 12.4 & 0 \\ 0 & 9.5 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} .58 & .58 & .58 & 0 & 0 \\ 0 & 0 & 0 & .71 & .71 \end{bmatrix}}_{V^T}$$

To perform SVD in python (scipy) use scipy.linalg.svd
http://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.svd.html

Dimensionality Reduction with SVD

- Procedure
  - Perform SVD on M. Retain the top-*k* (largest) singular values in Σ and set the remainder to zero.
  - Let us denote the diagonal matrix produced by the previous step by Σ_k
  - The *k*-dimensional approximation (projection) M_k of M is then given by
    - $M_k = U \Sigma_k V^T$

Reason

- The *k*-dimensional matrix M_k that minimises the Frobenius norm $||M-M_k||$ is given by the matrix M_k computed as described in the previous slide
- Frobenius norm
  - Extension of the vector L2 norm to matrices
  - Frobenius norm of a matrix M is given by $\sqrt{\sum_{ij} \mathbf{M}_{ij}{}^2}$

Proof

- By performing SVD on M, let
  - M = PQR^T

$$m_{ij} = \sum_k \sum_\ell p_{ik} q_{k\ell} r_{\ell j} \qquad \|M\|^2 = \sum_i \sum_j (m_{ij})^2 = \sum_i \sum_j \Big(\sum_k \sum_\ell p_{ik} q_{k\ell} r_{\ell j} \Big)^2$$

$$\Big(\sum_k \sum_\ell p_{ik} q_{k\ell} r_{\ell j} \Big)^2 = \sum_k \sum_\ell \sum_m \sum_n p_{ik} q_{k\ell} r_{\ell j} p_{in} q_{nm} r_{mj}$$

$$\|M\|^2 = \sum_i \sum_j \sum_k \sum_\ell \sum_n \sum_m p_{ik} q_{k\ell} r_{\ell j} p_{in} q_{nm} r_{mj}$$

$$\|M\|^2 = \sum_i \sum_j \sum_k \sum_n p_{ik} q_{kk} r_{kj} p_{in} q_{nn} r_{nj}$$

$$\|M\|^2 = \sum_j \sum_k q_{kk} r_{kj} q_{kk} r_{kj}$$

Much easier proof exists if you use
the trace of a matrix and its properties

$$\|M\|^2 = \sum_k (q_{kk})^2$$

Proof using Trace

$$\mathbf{M} = \mathbf{P}\mathbf{Q}\mathbf{R}^\top$$

$$||\mathbf{M}||_2^2 = \operatorname{tr}(\mathbf{M}^\top \mathbf{M})$$

$$||\mathbf{M}||_2^2 = \operatorname{tr}(\mathbf{R}\mathbf{Q}\mathbf{P}^\top \mathbf{P}\mathbf{Q}\mathbf{R}^\top)$$

$$||\mathbf{M}||_2^2 = \operatorname{tr}(\mathbf{R}\mathbf{Q}\mathbf{Q}\mathbf{R}^\top)$$

$$||\mathbf{M}||_2^2 = \operatorname{tr}(\mathbf{R}\mathbf{Q}^2\mathbf{R}^\top)$$

$$||\mathbf{M}||_2^2 = \operatorname{tr}(\mathbf{R}^\top\mathbf{R}\mathbf{Q}^2)$$

$$||\mathbf{M}||_2^2 = \operatorname{tr}(\mathbf{Q}^2)$$

SVD and Approximation Error

$$\mathbf{M} = \mathbf{P} \begin{bmatrix} \sigma_1 & & & & & \\ & \ddots & & & & \\ & & \sigma_k & & & \\ & & & \sigma_{k+1} & & \\ & & & & \ddots & \\ & & & & & \sigma_n \end{bmatrix} \mathbf{R}^\top$$

If M_k is the matrix with (k+1) and above singular values set to zero
(k-th rank approximation)

$$\mathbf{M}_k = \mathbf{P} \begin{bmatrix} \sigma_1 & & & & & \\ & \ddots & & & & \\ & & \sigma_k & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{bmatrix} \mathbf{R}^\top$$

SVD and Approximation error

- Then, the approximation error, $||M - M_k||^2$ becomes

$$||\mathbf{M} - \mathbf{M}_k||^2 = \mathrm{tr}\left(\mathbf{P} \begin{bmatrix} 0 & & & & & \\ & \ddots & & & & \\ & & 0 & & & \\ & & & \sigma_{k+1} & & \\ & & & & \ddots & \\ & & & & & \sigma_n \end{bmatrix} \mathbf{R}^\top \right)$$

$$||\mathbf{M} - \mathbf{M}_k||^2 = \sigma_{k+1}^2 + \ldots + \sigma_n^2$$

If we want to minimize this error, then we must select the largest singular values for the first 1…k positions!

Applications of SVD

- Latent Semantic Analysis
  - words vs. document matrix
  - Find similar words (query expansion)
  - Find similar documents (similarity search)
- Recommendation Systems
  - users vs. items/products
  - Recommend similar products to users

Two uses of SVD

- SVD for dimensionality reduction
  - Compute $M = U\Sigma V^T$
  - Get the largest *k* singular values from Σ to construct a diagonal matrix Σ_k
  - Get the corresponding left singular vectors from U to construct a matrix U_k
  - Reduce the number of columns of M to *k* to construct the matrix M_k
    - $M_k = U_k\Sigma_k$

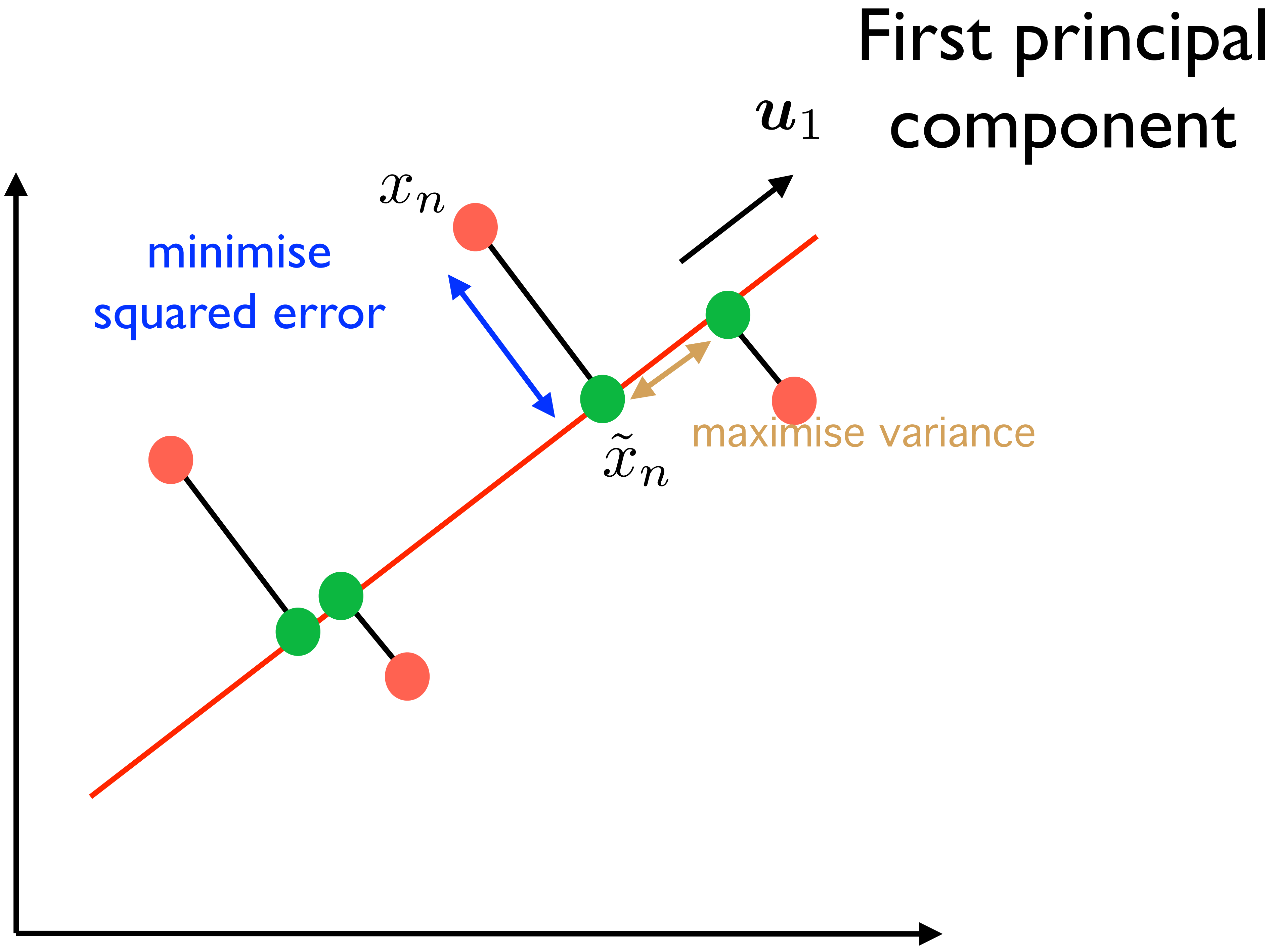
Two uses of SVD

- SVD to increase the density of a matrix
  - Compute $M = U\Sigma V^T$
  - Get the largest *k* singular values from Σ to construct a diagonal matrix Σ_k
  - Get the corresponding left singular vectors from U to construct a matrix U_k
  - Get the corresponding right singular vectors from U to construct a matrix V_k
  - Reproduce a dense version of M, M_k
    - $M_k = U_k \Sigma_k V_k^T$
    - Lesser number of non-zero values in M_k
    - However, we end up with negative values in M_k even though M is a matrix with all non-negative values!

Principal Component Analysis

- We would like to *project* our *high dimensional* data points to a *low dimensional* space by *preserving the geometric properties* in the original space as much as possible
- Two ways to do this:
  - Maximise the variance of the projected data
  - Linear projection that minimises the average projection cost (e.g. sum of squared Euclidean distance between original and projected points)
- PCA is also known as the *Karhunen-Loève Transform*

Idea


First principal component
$\boldsymbol{u}_1$
x_n
minimise squared error
$\tilde{x}_n$
maximise variance

Maximum Variance Formulation

- Problem
  - Given D dimensional N data points $\{\mathbf{x}_n\}$, where n=1,...,N, we must project those into an M<D dimensional space
  - M is given
- Let us consider the case M=1 (one-dimensional projection)
- The projection direction is given by the unit vector $\mathbf{u}_1$
  - $\mathbf{u}_1^\top \mathbf{u}_1 = 1$
- $\tilde{x}_n = \boldsymbol{u}_1{}^\top \boldsymbol{x}_n$

Maximum Variance Formulation

- Mean of the data points

$$\bar{\boldsymbol{x}} = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}_n$$

- Variance of the projected data

$$\frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{u}_1{}^\top\boldsymbol{x}_n - \boldsymbol{u}_1{}^\top\bar{\boldsymbol{x}})^2 = \boldsymbol{u}_1{}^\top\mathbf{S}\boldsymbol{u}_1$$

*An unbiased estimator of variance uses (N-1) instead of N.
But this does not matter because we are only interested in the maximisation of the variance.

- S is the covariance matrix given by

$$\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{x}_n - \bar{\boldsymbol{x}})(\boldsymbol{x}_n - \bar{\boldsymbol{x}})^\top$$

Maximum Variance Formulation

- We must maximize the variance subjected to the normalization constraint on **u**$_1$

Lagrange multiplier method

$$L(\boldsymbol{u}_1, \lambda_1) = \boldsymbol{u}_1{}^\top \mathbf{S} \boldsymbol{u}_1 + \lambda_1 (1 - \boldsymbol{u}_1{}^\top \boldsymbol{u}_1)$$

$$\frac{\partial L(\boldsymbol{u}_1, \lambda_1)}{\partial \boldsymbol{u}_1} = 0 \quad \Rightarrow \quad \mathbf{S}\boldsymbol{u}_1 = \lambda_1 \boldsymbol{u}_1$$

$$\boldsymbol{u}_1{}^\top \mathbf{S} \boldsymbol{u}_1 = \lambda_1$$

This is variance!

u$_1$ is the eigenvector of S that corresponds to the largest eigenvalue of S

PCA Algorithm

- INPUT
  - D dimensional N data points $\{x_n\}$, where $n=1,\ldots,N$
  - Dimensionality M
- Procedure
  - Compute the covariance matrix S for the dataset
  - Compute the first M eigenvectors of S
- return the computed eigenvectors

A Word on Complexity

- Eigenvalue decomposition of a DxD matrix is O(D^3)
- However, we only need the largest M eigenvectors of S
- This can be computed efficiently using truncated methods such as the power-iteration method in O(MD^2)
- Reference
    - Golub & Van Loan, *Matrix Computations*, John Hopkins University Press, 1996.

Computational Remarks (1/2)

- The following methods for computing PCA are equivalent (i.e. gives the same principal components)

1. Compute the sum of **squared** Euclidean *distance* between original and projected points and minimise it.
2. Compute the **pairwise squared** Euclidean *distance* between projected points and maximise it.
3. Compute the *variance* of the projected points on the projection line and maximise it.

Computational Remarks (2/2)

- Let us show that (2) and (3) are equivalent

$$
\begin{aligned}
(3) &= \sum_{i=1}^{N} (\boldsymbol{x}_i \boldsymbol{u} - \bar{\boldsymbol{x}}^\top \boldsymbol{u})^2 \\
&= \sum_{i=1}^{N} \left(\left(\boldsymbol{x}_i - \frac{1}{N} \sum_{j=1}^{N} \boldsymbol{x}_j \right)^\top \boldsymbol{u} \right)^2 \\
&= \sum_{i=1}^{N} \sum_{j=1}^{N} \left((\boldsymbol{x}_i - \boldsymbol{x}_j)^\top \boldsymbol{u} \right)^2 \\
&= \sum_{i=1}^{N} \sum_{j=1}^{N} (\boldsymbol{x}_i{}^\top \boldsymbol{u} - \boldsymbol{x}_j{}^\top \boldsymbol{u})^2 \\
&= (2)
\end{aligned}
$$

Using (2) for computing PCA is usually a bad idea when the dataset is large because the pairwise combinations grow quadratically with the number of data points in the dataset. But could be helpful for small examples because we do not have to compute the mean.

References

- Mining of Massive Datasets
  - http://infolab.stanford.edu/~ullman/mmds.html#original
  - Chapter 11 on Dimensionality Reduction (SVD)
- Pattern Recognition and Machine Learning
  - Section 1.1 (overfitting)
  - Section 1.4 (curse of dimensionality)
  - Page 561 onwards: PCA