DS 503: Advanced Data Analytics

Week 2: Projection Techniques

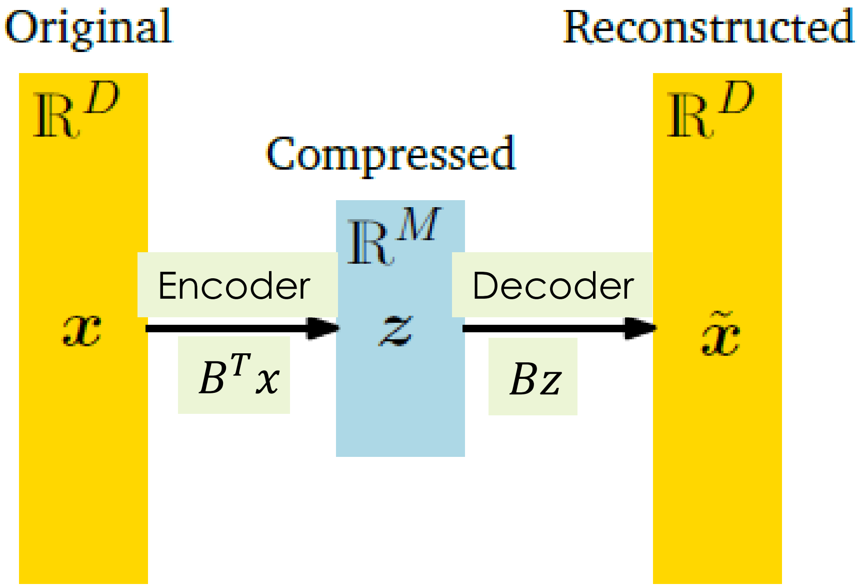
Gagan Raj Gupta

How to deal with high-dimensions?

- Project data to a low-dimensional space
- Fidelity: While doing so, don’t disturb the relative distances
- Where is this used?
  - Principal component analysis
  - Nearest Neighbor Search
  - Clustering
  - Text Embeddings
  - Graph Embeddings
  - Image manifolds

Problem statement

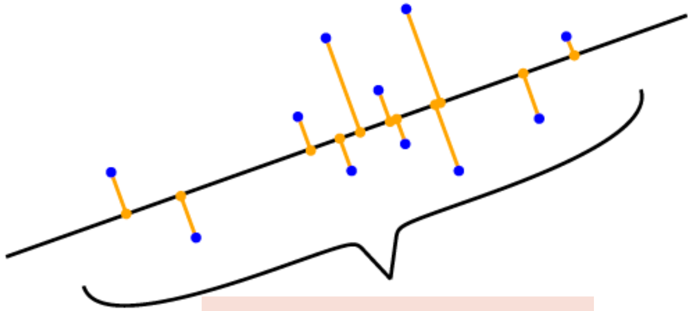
- $\boldsymbol{D}$ **x** $\boldsymbol{n}$ data matrix A (D rows and $\boldsymbol{n}$ columns)
  - Each column is a D-dimensional vector
  - Columns are normalized (mean = 0)
- Assume there is a projection matrix B, such that
  - $z = B^T x \in R^M$
  - $B = [b_1, b_2, \dots, b_M]$ has orthogonal columns
- B is the best-fit M-dim. subspace S_M for rows of A
  - Minimize reconstruction error
  - Minimize sum of squared distances from A_i to S_M



Minimize:

$$||x - \tilde{x}||^2$$

Best fit subspaces and Maximizing Information



Best Line (1-d)

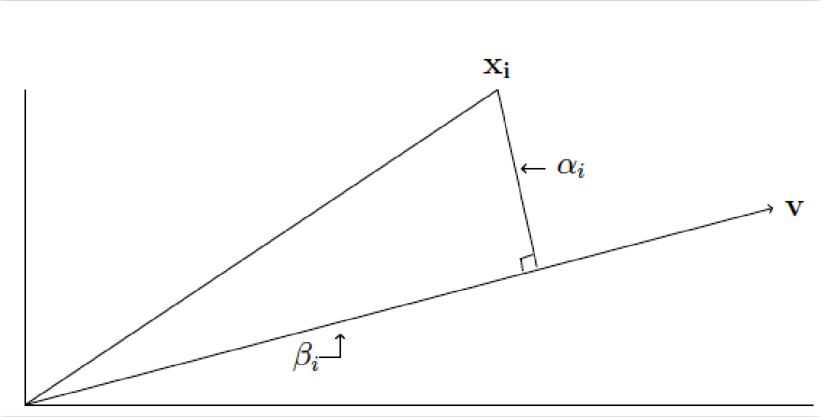
- Let's begin with the following question.
  - Find a direction in which the data ($\boldsymbol{D}$ **x** $\boldsymbol{n}$ matrix) has maximum information
  - Maximize $|Av_1|$, such that $|v_1| = 1$
  - This is also equivalent to Minimizing the sum of squared distances to the point nearest to the line.
- By successively solving the above problem, we can find the best-fit k-dimensional subspaces
  - Which preserve the maximum possible information (by maximizing projections)
  - Or equivalently, minimize the sum of squared distances of the vectors to the subspace
- When k=r, the rank of A, we get SVD

Projections, Distances and Data Variance

- Minimizing distance = maximizing projection
  - $||\boldsymbol{x}||_2^2 = (projection)^2 + (distance\ to\ line)^2$
- SVD: Find best fit 1-dimensional line
  - u_1 = unit vector along the best fit line
  - x_i= i-th column of A, length of its projection: $|\langle \boldsymbol{x}_i, u\rangle|$
- Sum of squared projection lengths: $||A^T u||_2^2$
- **First singular vector**:

$$\boldsymbol{u}_1 = \arg \max_{||u||_2=1} ||A^T u||_2$$

- If there are ties, break arbitrarily
  - $\sigma_1(A) = ||A^T \boldsymbol{u}_1||_2$ is the **first singular value**



Variance of z_1 of $z \in R^M$

$$V_1 \coloneqq V[z_1] = \frac{1}{N}\sum_{n=1}^{N} z_{1n}^2$$

$$z_{1n} = b_1^T x_n$$

$$V_1 = b_1^T\ S\ b_1$$

Where S is the data-covariance matrix

$$\boldsymbol{S} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^\top .$$

Maximizing Variance using PCA

Optimization Problem

$$\max_{\boldsymbol{b}_1} \boldsymbol{b}_1^\top \boldsymbol{S} \boldsymbol{b}_1$$

$$\text{subject to } \|\boldsymbol{b}_1\|^2 = 1\,.$$

Encoder

$$z_{n1} = b_1^T x_n \in R$$

Decoder

$$\tilde{x}_n = b_1 z_{n1} = b_1 b_1^T x_n \in R^D$$

Lagrangian

$$\mathfrak{L}(\boldsymbol{b}_1, \lambda) = \boldsymbol{b}_1^\top \boldsymbol{S} \boldsymbol{b}_1 + \lambda_1 (1 - \boldsymbol{b}_1^\top \boldsymbol{b}_1)$$

Differentiating

$$\frac{\partial \mathfrak{L}}{\partial \boldsymbol{b}_1} = 2\boldsymbol{b}_1^\top \boldsymbol{S} - 2\lambda_1 \boldsymbol{b}_1^\top\,, \qquad \frac{\partial \mathfrak{L}}{\partial \lambda_1} = 1 - \boldsymbol{b}_1^\top \boldsymbol{b}_1\,,$$

$$\boldsymbol{S}\boldsymbol{b}_1 = \lambda_1 \boldsymbol{b}_1\,,$$

$$\boldsymbol{b}_1^\top \boldsymbol{b}_1 = 1\,.$$

b_1 is an eigenvector of S
λ_1 is the eigenvalue

$$V_1 = \boldsymbol{b}_1^\top \boldsymbol{S} \boldsymbol{b}_1 = \lambda_1 \boldsymbol{b}_1^\top \boldsymbol{b}_1 = \lambda_1\,,$$

- Variance of the data projected onto a one-dimensional subspace equals the **eigenvalue** that is associated with the **basis vector b1** that spans this subspace.
- To maximize the variance of the low-dimensional code, we choose the basis vector associated with the **largest eigenvalue** principal component of the **data covariance matrix**.
- This eigenvector is called the first *principal component*

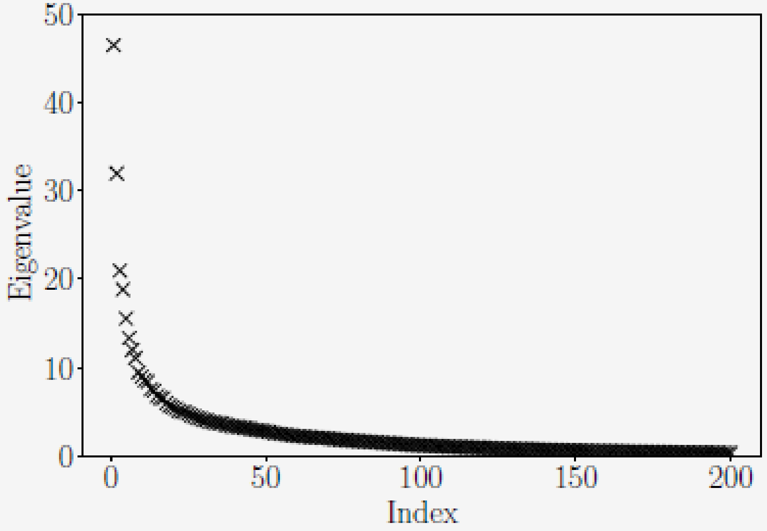
General Algorithm

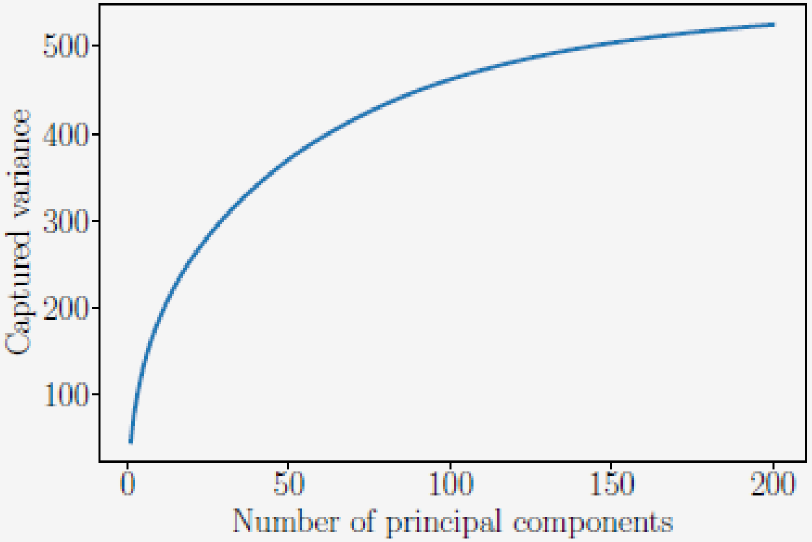
- In general, the largest M (orthonormal) eigenvectors of the data-covariance matrix span the best M dimensional subspace for A
- They also capture the variance equal to the sum of largest M eigenvalues
- We can iteratively compute the largest eigenvalue/eigenvector of the covariance matrix ‘S’ and continue till the sum of M eigenvalues captures a large fraction of the trace of S

$$V_M = \sum_{m=1}^{M} \lambda_m$$

$$1 - \frac{V_M}{V_D}.$$

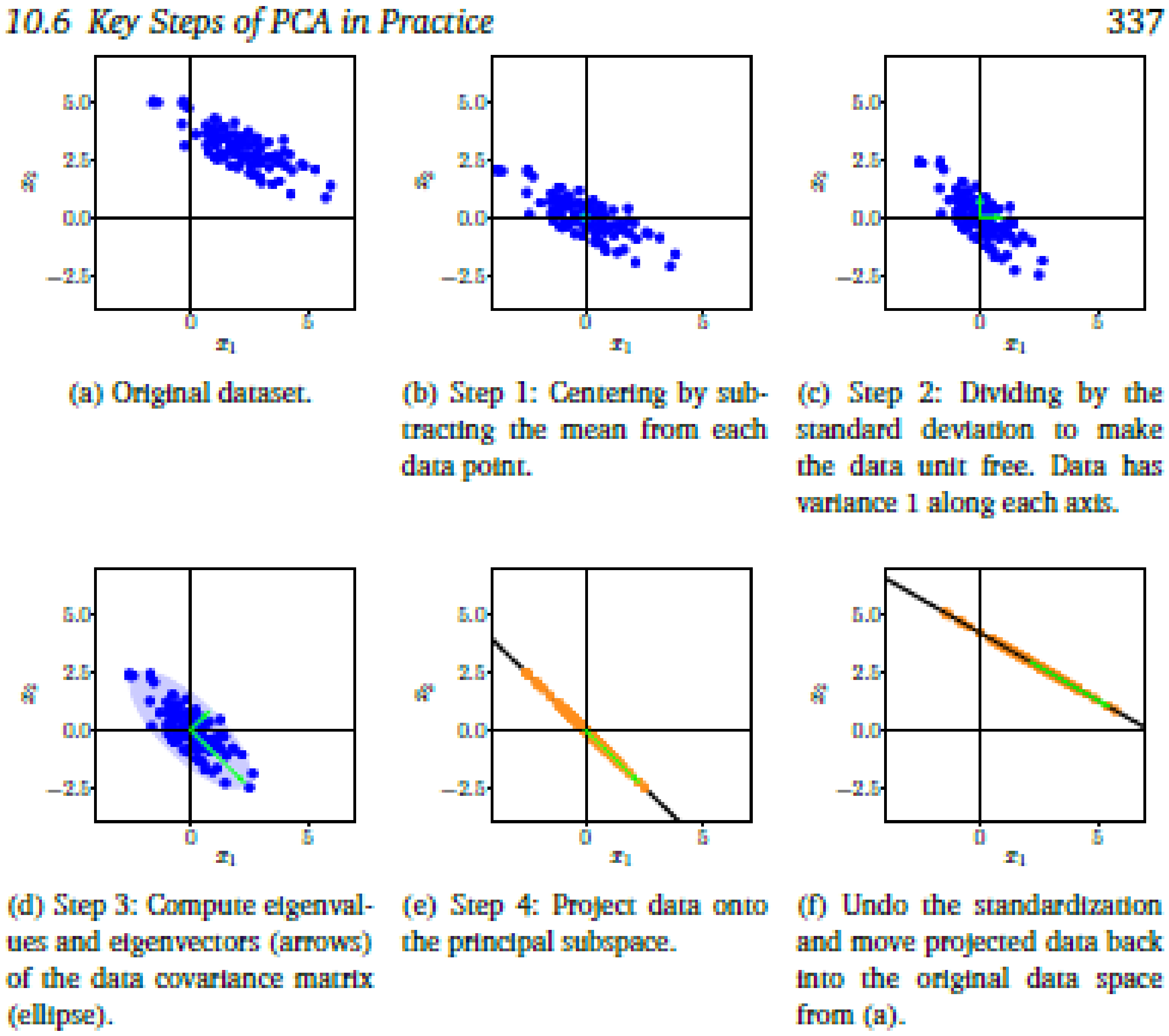
Stop when this become small





Key Steps of PCA in Practice

- Centering of the data
- Normalization
- Eigen decomposition of the Covariance Matrix
- Projection
- Mapping back to the original data space



10.6 Key Steps of PCA in Practice 337

(a) Original dataset.

(b) Step 1: Centering by subtracting the mean from each data point.

(c) Step 2: Dividing by the standard deviation to make the data unit free. Data has variance 1 along each axis.

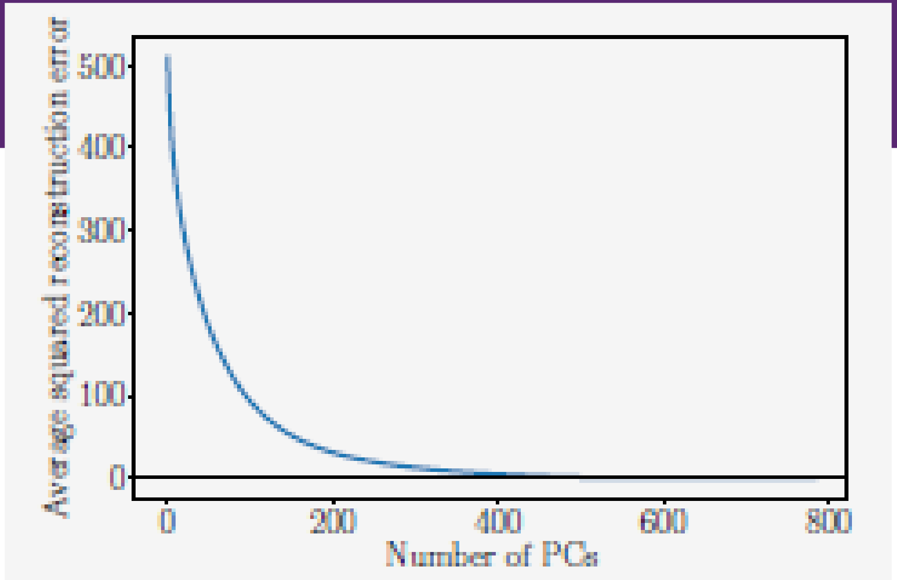
(d) Step 3: Compute eigenvalues and eigenvectors (arrows) of the data covariance matrix (ellipse).

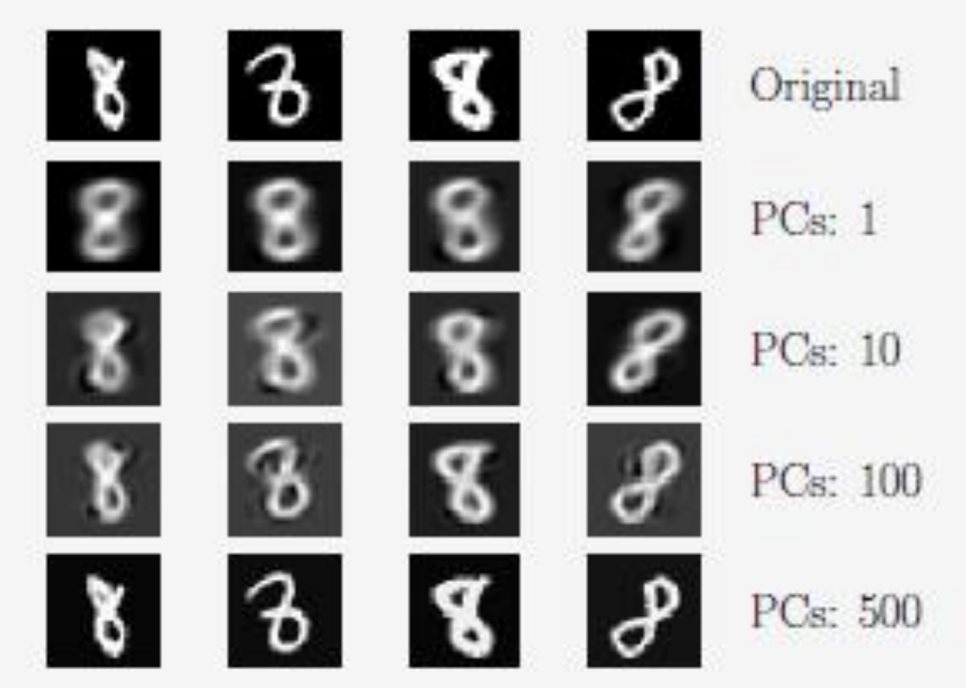
(e) Step 4: Project data onto the principal subspace.

(f) Undo the standardization and move projected data back into the original data space from (a).

MNIST Digits: reconstruction

- MNIST dataset contains of handwritten digits 0 to 9
- Each digit is an image of 28x28 pixels, that is equivalent to a vector of dimension 784
- The picture on the right is performing PCA on the 5389 samples of “8” from the dataset
- We check the reconstruction error curve to decide how many components to keep





Variants of PCA

- **Sparse PCA** : Add a L1 regularization term to remove features with small coefficients. Find sparse low rank matrices B and C so that A ≈ BC
- **Non-negative Matrix Factorization:** Find non-negative matrices U and V so that A ≈ UV
- **Kernel PCA** : Account for a non-linear relationship among features by using the “kernel trick”
- **Probabilistic PCA (PPCA):** Formulates PCA as a generative process which allows us to simulate new data

Application Areas:

- **Facial Feature Extraction**

Find a few basic faces so that every face can be expressed as a combination

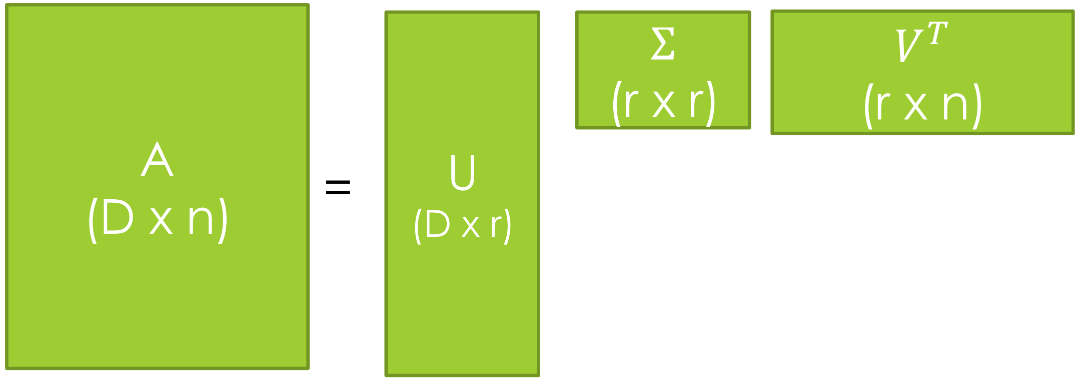
- **Gene Expressions**

Find out which genes are playing a role in a particular phenomenon

- **Text Mining and Document Classification**

Combine the various topics in a document

Properties of SVD



- SVD: $A = U\Sigma V^T$
- $\Sigma =$ Diagonal matrix (positive real entries σ_{ii})
- U, V: orthonormal columns:
  - $\boldsymbol{u}_1, \dots, \boldsymbol{u}_r \in \mathbb{R}^D$ (directions that maximize projections of x_i)
  - $\boldsymbol{v}_1, \dots, \boldsymbol{v}_r \in \mathbb{R}^n$ (projections of x_i on u_i)
  - $\langle \boldsymbol{u}_i, \boldsymbol{u}_j \rangle = \delta_{ij}, \langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle = \delta_{ij}$
- $A = \sum_i \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$
- Thus, any matrix can be represented as sum of “r” rank-1 matrices

Singular Values vs. Eigenvalues

- If A is a square matrix:
  - Vector $\boldsymbol{v}$ such that $A\boldsymbol{v} = \lambda\boldsymbol{v}$; is an eigenvector with eigenvalue = λ
  - For symmetric real matrices, A, $\boldsymbol{v}$'s are orthonormal
    - A can be expressed as: $A = \mathrm{V}\Sigma V^T = U\Sigma U^T$
    - V's columns are eigenvectors of A
  - Diagonal entries of Σ are eigenvalues $\lambda_1, \ldots, \lambda_n$
- SVD is defined for all matrices (not just square)
  - Orthogonality of singular vectors is automatic

$$A\boldsymbol{v}_i = \sigma_i \boldsymbol{u}_i \text{ and } A^T\boldsymbol{u}_i = \sigma_i \boldsymbol{v}_i \text{ (will show)}$$

$$AA^T u_i = \sigma_i^2 \boldsymbol{u}_i \Rightarrow u_i's \text{ are eigenvectors of } AA^T \text{ ((N-1) times sample covariance matrix)}$$

SVD: Greedy Construction

- Find best fit 1-dimensional line, repeat r times (until projection is 0)
- **Second singular vector and value:**

$$\boldsymbol{u}_2 = \arg \max_{u \perp u_1, \left|\left|u\right|\right|_2 = 1} \left|\left|A^{\boldsymbol{T}} u\right|\right|_2$$

$$\sigma_2(A) = \left|\left|A\boldsymbol{u}_2\right|\right|_2$$

- **k-th singular vector and value:**

$$\boldsymbol{u}_k = \arg \max_{u \perp \boldsymbol{u}_1, \dots \boldsymbol{u}_{k-1}, \left|\left|u\right|\right|_2 = 1} \left|\left|A^{\boldsymbol{T}} u\right|\right|_2$$

$$\sigma_k(A) = \left|\left|A^T \, \boldsymbol{u}_k\right|\right|_2$$

- We can show that:$(\boldsymbol{u}_1, \boldsymbol{u}_2, \dots, \boldsymbol{u}_k)$ is best-fit subspace

Best low rank approximations of a Matrix

- Suppose I have to find the best “k” rank approximation A_k matrix.

  Let $A_k = \sigma_1 u_1 v_1^T + \ ... + \sigma_k u_k v_k^T$
  Eckart-Young: If B has rank k then $\left||A - B|\right| \geq |A - A||_k$

- This result has been proven for multiple norms

- Spectral: $\left||A|\right|_2 = \max \frac{||Ax||}{||x||} = \sigma_1$

- Frobenius: $\left||A|\right|_F^2 = \sigma_1^2 + \cdots + \sigma_r^2 = \sum_{i,j} \left|a_{i,j}\right|^2 = trace\ of\ AA^T$

- Nuclear: $\left||A_N|\right| = \sigma_1 + \ ... + \sigma_r$ (the trace norm)

Applications of low rank approximations

- Principal component analysis (fitting a hyperplane to data)
- Model reduction in analyzing physical systems
- Fast algorithms in scientific computing
- PageRank and other spectral methods in data analysis
- Diffusion geometry and manifold learning
- Many, many more …

Computing the SVD: Power iteration method

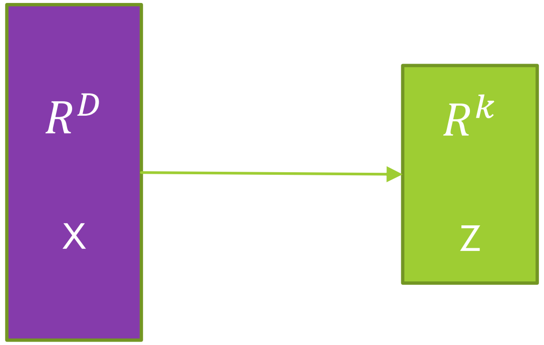
- Begin with a random vector x_0 that is not in the null space of S = AA^T
- Follow the iteration $\mathrm{x}_{\mathrm{k+1}} = \frac{S\, x_k}{||S\, x_k||}$
- This converges to the eigenvector associated with the largest eigenvalue of S
- Also used in the page-rank algorithm for ranking web-pages based on their hyperlinks
- Issues: S can be very large
- Alternative 1: If N << D, then we can instead work with the matrix $A^T A$ which also has the same eigenvalues (square of singular values).
- This can happen, for example, when we are dealing with large size images with millions of pixels.

SVD works well for small matrices (m, n< 5000) or sparse matrices. The function for computing SVD is not so easy to write.

JL Lemma and Random Projections

- Suppose $x_1, x_2, \ldots, x_n$ are any n points in R^D and $k \geq \frac{8 \ln n}{\epsilon^2}$.
- ∃ a distance preserving projection J from R^D to R^k which works ∀ pairwise distances:
    - $\boxed{(1-\epsilon)\left|\left|x_i - x_j\right|\right|^2 \leq \left|\left|Jx_i - Jx_j\right|\right|^2 \leq (1+\epsilon)\left|\left|x_i - x_j\right|\right|^2}$
- There are multiple proofs and one of the proof shows that w.h.p. a random projection (kxD) is very likely to keep the n points apart.
- Project each $\boldsymbol{x} \in \boldsymbol{A}$ onto $f(\boldsymbol{x})$, where $f: \mathbb{R}^{\boldsymbol{D}} \rightarrow \mathbb{R}^{\boldsymbol{k}}$
- Pick $\boldsymbol{k}$ vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{\boldsymbol{k}}$ i.i.d: $\boldsymbol{u}_i \sim N_D(0^D, 1)$

$$f(\boldsymbol{v}) = (\langle \boldsymbol{u}_1, \boldsymbol{v} \rangle, \ldots, \langle \boldsymbol{u}_{\boldsymbol{k}}, \boldsymbol{v} \rangle)$$



    - Since the k vectors were Gaussian, the projections are also Gaussian.
    - Application of Gaussian Annulus theorem allows to bound the deviation in the distances

Applications

- Suppose $x_1, x_2, \ldots, x_n$ are any n points in R^D, where D is very large. Tasks:
- Suppose the points almost live on a linear subspace of (small) dimension *k*.
  - Find a basis for the “best” subspace. (Principal component analysis.)
- Given *k*, find the subset of *k* vectors with maximal spanning volume.
- Suppose the points almost live on a low-dimensional nonlinear manifold.

  Find a parameterization of the manifold.
- Given *k*, find for each vector $\boldsymbol{x_i}$, its *k* closest neighbors.
- Partition the points into clusters.

Note: Some of these don’t have well-defined solutions and some are combinatorially hard. If we can embed the points in a low-dimensional subspace, while preserving distance approximately, then a variety of algorithms for solving these problems become possible.

Application: Nearest Neighbors Search

Goal: Given a database of n points in R^D where n and D are usually large. Query points in R^D. Queries should be fast.

- If the database has n1 points and n2 queries are expected during the lifetime of the algorithm, take n = n1 + n2
- Project database (using random vectors) to a k-dimensional space
- Store projections in an efficient data structure (more in next lecture)
- On receiving a query, project the query to the same subspace and compute nearby database points.
- The JL Lemma says that with high probability this will yield the right answer whatever the query

Linear Regression (Problem setup)

- We are given observations **(x,** y)
  - Let us build a linear model to predict y from the observations **x** s. t. $\mathrm{w}_0 + \sum_{i=1}^{d} w_i x_i = y$
- Compute the “best” solution to the model $Xw = y$ that minimizes SSE (sum of squared distances)
- This problem is very similar to solving $Ax = b$, a system of linear equations
- In most cases, there may be no exact solution to this problem
- But, there are many approaches we can take to get the “least squares” solution
  - This minimizes the sum of square errors $||b - A\hat{x}||^2$

<u>Normal Equations</u>

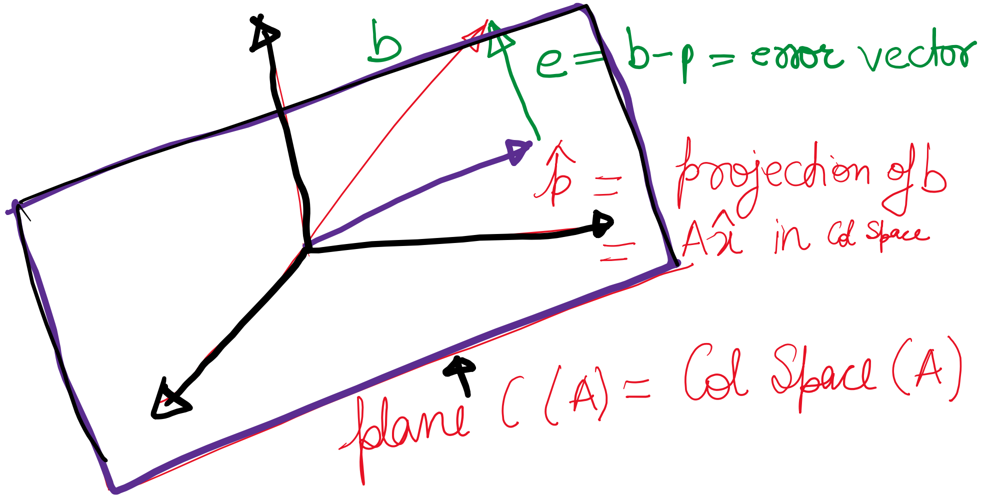
Since $(b - A\hat{x})$ is perpendicular to all vectors Ax in the column space, $(Ax)^T(b - A\hat{x}) = \mathrm{x}^\mathrm{T}\mathrm{A}^\mathrm{T}(\mathrm{b} - A\hat{x}) = 0$

Normal Equation for solving $\hat{x}$ **:** $A^T A\hat{x} = A^T b$

Least squares sol to Ax=b: $\hat{x} = (A^T A)^{-1} A^T b$

Projection of b onto Col(A): $p = A\hat{x} = A(A^T A)^{-1} A^T b$

Projection matrix that multiplies b to give p: $P = A(A^T A)^{-1} A^T$

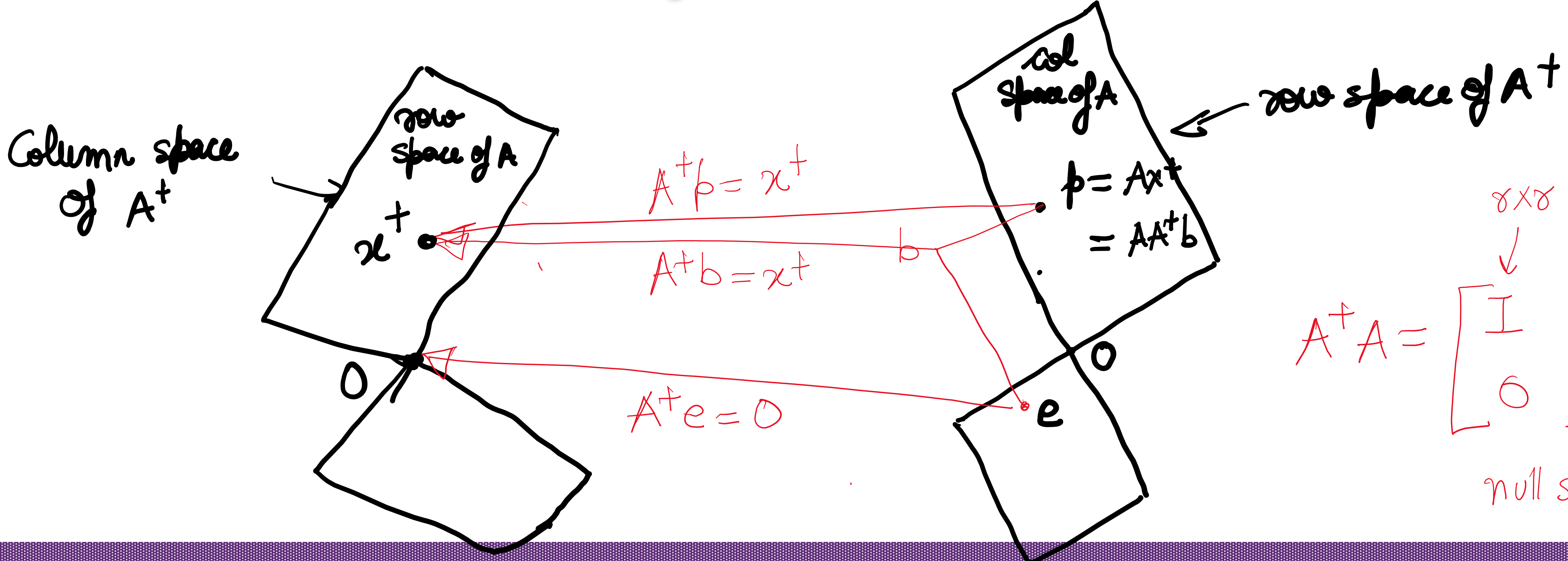


Pseudo Inverse Method

- Pseudo Inverse Method: $\hat{x} = A^+ b$
  - If A has independent columns, $A^+ = (A^T A)^{-1} A^T$
  - If A has independent rows, $A^+ = A^T (A A^T)^{-1}$
  - Pseudo inverse can be computed using SVD: $A^+ = V \Sigma^+ U^T$
  - Σ^+ contains the inverse of all non-zero diagonal elements in Σ

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\Sigma^+ = \begin{bmatrix} 1/\sigma_1 & 0 & 0 \\ 0 & 1/\sigma_2 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$



r×r

$$A^+A = \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix}$$

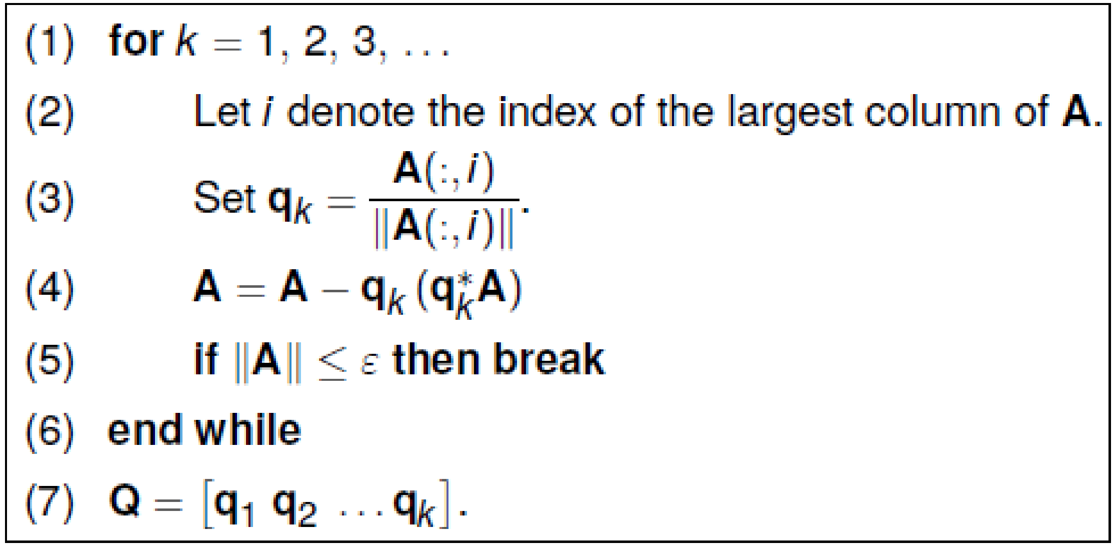
null space

QR (Gram-Schmidt) Method

- Decompose $A = QR$
  - Where Q is an orthogonal matrix, and R is a triangular matrix
- Then, $A^T A = R^T Q^T Q R = R^T R$ and the normal equation $A^T A \hat{x} = \mathrm{A}^{\mathrm{T}}\mathrm{b}$, can be solved as
  - $\hat{\boldsymbol{x}} = (R^T R)^{-1} R^T Q^T \boldsymbol{b} = \boldsymbol{R}^{-\mathbf{1}} \boldsymbol{Q}^{\boldsymbol{T}} \boldsymbol{b}$
- This is computationally efficient to solve

Alternatives to compute Low Rank Approximations

- Let A be an mxn matrix of low numerical rank
- Suppose that you can't afford to compute the full SVD or you don't have a good implementation
- How can you compute a low-rank approximation to A?
  - Gram-Schmidt: Keep reducing a rank-1 component from A
    - Complexity: O(mnk)
  - Krylov Methods: Restrict the matrix A to the k-dimensional "Krylov subspace"
    - Span $(r. Ar, A^2r, \ldots, A^{k-1}r)$
    - Compute eigen-decomposition of resulting matrix



```
for $k = 1, 2, 3, \ldots$
    Let $i$ denote the index of the largest column of $\mathbf{A}$.
    Set $\mathbf{q}_k = \frac{\mathbf{A}(:,i)}{\|\mathbf{A}(:,i)\|}$.
    $\mathbf{A} = \mathbf{A} - \mathbf{q}_k\,(\mathbf{q}_k^*\mathbf{A})$
    if $\|\mathbf{A}\| \le \varepsilon$ then break
end while
$\mathbf{Q} = [\mathbf{q}_1\ \mathbf{q}_2\ \ldots \mathbf{q}_k]$.
```

Each of these approximations result in a factorization of the form

$$A \approx Q\, Q^T A$$

Where Q is an approximate orthonormal basis for the column space of A

Randomized Low Rank Approximations

Range Finding (Basis) Problem: Given an m x n matrix A and an integer k < min(m,n). Find an orthonormal m x k matrix Q such that $A \approx Q\, Q^T A$

Solving the primitive problem via randomized sampling — intuition:

1. Draw Gaussian random vectors $g_1, g_2, \ldots, g_k \in R^n$
2. Form "sample" vectors $y_1 = Ag_1, y_2 = Ag_2, \ldots, y_k = Ag_k \in R^m$
3. Form orthonormal vectors $q_1, q_2, \ldots, q_k \in R^m$ such that

   Span($q_1, q_2, \ldots, q_k$) = Span($y_1, y_2, \ldots, y_k$)

   For instance, Gram-Schmidt can be used — pivoting is rarely required.

If A has exact rank k, then Span$\{q_j\}_{j=1}^{k}$= Range(A) with probability 1.

Randomized SVD

- **Goal:** Given an *m x n* matrix **A**, compute an approximate rank-*k* SVD $\boxed{A \approx U\Sigma V^T}$
- **Algorithm:**
- 1. Draw an *n x k* Gaussian random matrix **G**. G = randn(n,k)
- 2. Form the *m x k* sample matrix **Y** = **AG** Y = A * G
- 3. Form an *m x k* orthonormal matrix **Q** such that **Y** = **QR** [Q, ~] = qr(Y)
- 4. Form the *k x n* matrix **B** = $Q^T A$ B = Q’ * A
- 5. Compute the SVD of the small matrix **B**: **B** = $\widehat{U}\Sigma V^T$ [Uhat, Sigma, V] = svd(B,0)
- 6. Form the matrix **U** = $Q\widehat{U}$ U = Q * Uhat
- **Power iteration to improve the accuracy:** The computed factorization is close to optimally accurate when the singular values of **A** decay rapidly. When they do not, a small amount of power iteration should be incorporated, i.e. replace Step 2 by **Y =**$\left(AA^T\right)^t$**AG** for a small integer *t*, say *t* = 1 or *t* = 2.

Randomized Embeddings: “Fast” JL Transforms

- So far, the only randomized embedding we have described takes the form

$$\boxed{f : R^D \to R^k : x \to \frac{1}{\sqrt{k}} Gx}$$

where **G** is a matrix drawn from a Gaussian distribution. Evaluating *f*(**x**) costs *O*(*nk*).

- The cost can be reduced to *O*(*n* log *k*) or even less, by using *structured* random maps.
  - Subsampled Fourier transforms. (Or Hadamard transform / cosine transform / . . .)
  - Sparse random embeddings — pick matrix that consists mostly of zeros.
  - Chains of random Givens’ rotations.