

Unit 11

Principal Component Analysis

EL-GY 6143: INTRODUCTION TO MACHINE LEARNING

PROF. PEI LIU

Learning Objectives

- ❑ Identify cases to use dimensionality reduction
- ❑ Mathematically describe principal components representations of data
- ❑ Compute principal components via SVDs
- ❑ Compute PC components in python
- ❑ Add PCA transforms as a pre-processing step to classification and regression
- ❑ Implement low-rank transforms for recommender systems



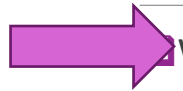
NYU

TANDON SCHOOL
OF ENGINEERING

2



Outline



Why dimensionality reduction?

- ☐ Principal components and directions of variance
- ☐ Approximation with PCs
- ☐ Computing PCs via the SVD
- ☐ Face example in python
- ☐ Training models from PCs
- ☐ Low rank approximations and recommender systems

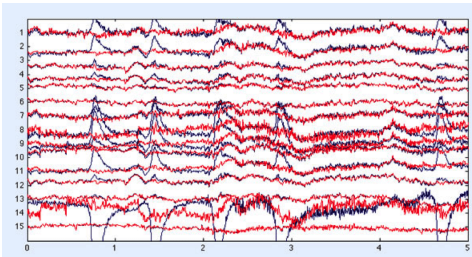


NYU

TANDON SCHOOL
OF ENGINEERING

High-Dimensional Data

- ❑ Many data sets have very high dimension
- ❑ Training can be difficult
 - Especially when number of samples is small
 - Classifier needs many parameters



EEG

Ex: 32 channels x 1 kHz x 10s



Face recognition with high-resolution images

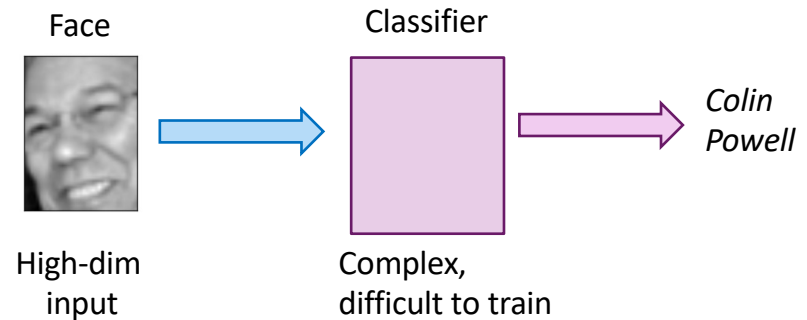


NYU

TANDON SCHOOL
OF ENGINEERING

Problems with High-Dimensions

- ❑ Consider face recognition
- ❑ Input is high-dimensional
 - Esp. for high resolution image
- ❑ Resulting classifier:
 - Requires many parameters
 - Difficult to train
 - Needs many samples
 - Computationally complex



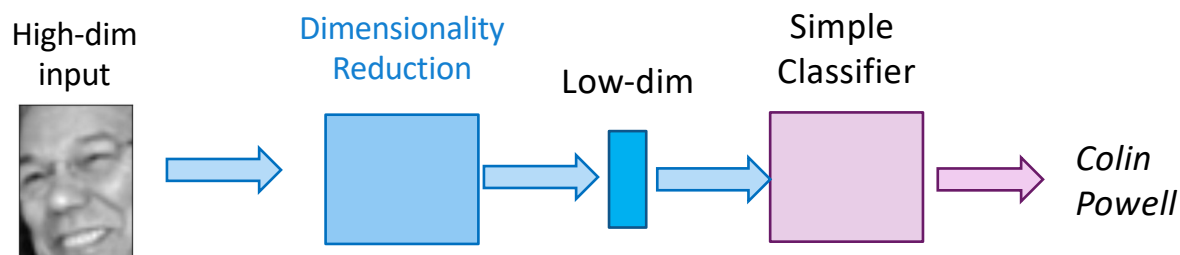
NYU

TANDON SCHOOL
OF ENGINEERING

5



Dimensionality Reduction



- ❑ Dimensionality reduction:
 - Reduce the input dimension to lower dimensional representation
- ❑ Can build simpler classifier
- ❑ Low-dimensional representational also good for:
 - Visualizing data
 - Clustering and other unsupervised tasks
 - Finding underlying structure of the data



NYU

TANDON SCHOOL
OF ENGINEERING

6



Outline

- ☐ Dimensionality reduction

-  ☒ Principal components and directions of variance

- ☐ Approximation with PCs

- ☐ Computing PCs via the SVD

- ☐ Face example in python

- ☐ Training models from PCs

- ☐ Low rank approximations and recommender systems



NYU

TANDON SCHOOL
OF ENGINEERING

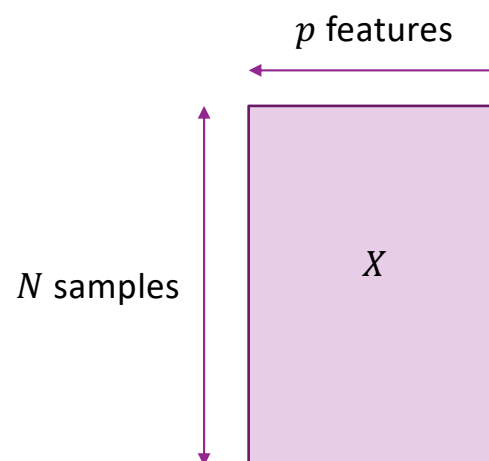
Data Definitions

- Given data: $\mathbf{x}_i, i = 1, \dots, N$
 - Each sample has p features: $\mathbf{x}_i = (x_{i1}, \dots, x_{ip})$
 - Represent as an $N \times p$ matrix

- Unsupervised learning
 - Samples do not have a label
 - Or we choose to ignore the label for now

- Dimension p is large

- How do we reduce the dimension?



Projections

□ PCA reduces dimensionality by “projecting” data to a lower dim subspace

□ **Projection:** Given vectors \mathbf{z} and \mathbf{v} , the projection of \mathbf{z} onto \mathbf{v} is:

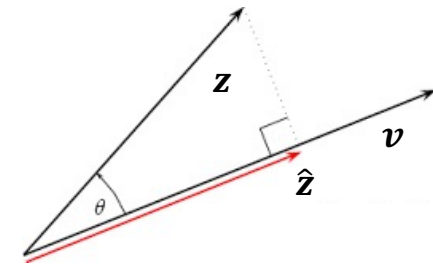
$$\hat{\mathbf{z}} = \text{Proj}_{\mathbf{v}}(\mathbf{z}) = \alpha \mathbf{v}, \quad \alpha = \frac{\mathbf{v}^T \mathbf{z}}{\mathbf{v}^T \mathbf{v}} = \frac{\|\mathbf{z}\|}{\|\mathbf{v}\|} \cos \theta$$

- α = coefficient of the projection

□ **Theorem:** $\text{Proj}_{\mathbf{v}}(\mathbf{z})$ is closest point in V to \mathbf{z} :

$$\hat{\mathbf{z}} = \arg \min_{\mathbf{w} \in V} \|\mathbf{z} - \mathbf{w}\|^2$$

- $V = \{\alpha \mathbf{v} | \alpha \in \mathbb{R}\}$ = vectors on the line spanned by \mathbf{v}



NYU

TANDON SCHOOL
OF ENGINEERING

Maximal Directional Variance

□ Given data: $x_i, i = 1, \dots, N$ and direction v with $\|v\| = 1$

□ Let $z_i = v^T x_i$ = coefficient of the projection of x_i onto v

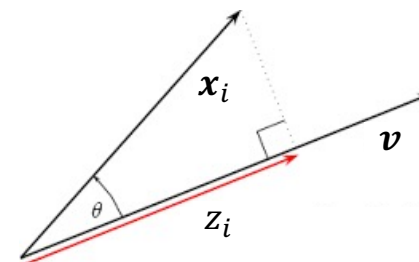
□ Sample mean and variance in direction v is :

- Sample mean $\bar{z} = \frac{1}{N} \sum_{i=1}^N z_i$
- Sample variance $s_z^2 = \frac{1}{N} \sum_{i=1}^N (z_i - \bar{z})^2$

Problem: Find the direction v that maximizes the variance s_z^2

□ Why?

- Captures the most variation of the data
- Provides the best vector for dimensionality reduction



Sample Covariance Matrix

□ Sample mean of the data: $\bar{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i$, and matrix $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix}$

□ Sample covariance matrix: Matrix \mathbf{Q} with components:

$$Q_{k\ell} = \frac{1}{N} \sum_{i=1}^N (x_{ik} - \bar{x}_k)(x_{i\ell} - \bar{x}_\ell)$$

- Covariance between feature k and ℓ in the dataset
- Matrix is $p \times p$

□ Define $\tilde{\mathbf{x}}_i = \mathbf{x}_i - \bar{\mathbf{x}}$, data with sample mean removed

□ Sample covariance is given by

$$\mathbf{Q} = \frac{1}{N} \sum_{i=1}^N (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T = \frac{1}{N} \begin{bmatrix} \tilde{\mathbf{x}}_1 & \tilde{\mathbf{x}}_2 & \cdots & \tilde{\mathbf{x}}_N \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{x}}_1^T \\ \tilde{\mathbf{x}}_2^T \\ \vdots \\ \tilde{\mathbf{x}}_N^T \end{bmatrix} = \frac{1}{N} \tilde{\mathbf{X}}^T \tilde{\mathbf{X}}$$



Sample Covariance and Directional Variance

□ Let $z_i = \mathbf{v}^T \mathbf{x}_i$ = coefficient of the projection of \mathbf{x}_i onto \mathbf{v}

□ Can compute the sample mean and variance of z_i from $\bar{\mathbf{x}}$ and \mathbf{Q}

□ **Sample mean** of the coefficients:

$$\bar{z} = \frac{1}{N} \sum_{i=1}^N z_i = \frac{1}{N} \sum_{i=1}^N \mathbf{v}^T \mathbf{x}_i = \mathbf{v}^T \left[\frac{1}{N} \sum_{i=1}^N \mathbf{x}_i \right] = \mathbf{v}^T \bar{\mathbf{x}}$$

□ **Sample variance** of the coefficients:

$$\begin{aligned} s_z^2 &= \frac{1}{N} \sum_{i=1}^N (z_i - \bar{z})^2 = \frac{1}{N} \sum_{i=1}^N (\mathbf{v}^T (\mathbf{x}_i - \bar{\mathbf{x}}))^2 \\ &= \frac{1}{N} [\mathbf{v}^T \tilde{\mathbf{x}}_1 \quad \mathbf{v}^T \tilde{\mathbf{x}}_2 \quad \cdots \quad \mathbf{v}^T \tilde{\mathbf{x}}_N] \begin{bmatrix} \tilde{\mathbf{x}}_1^T \mathbf{v} \\ \tilde{\mathbf{x}}_2^T \mathbf{v} \\ \vdots \\ \tilde{\mathbf{x}}_N^T \mathbf{v} \end{bmatrix} = \frac{1}{N} \mathbf{v}^T [\tilde{\mathbf{x}}_1 \quad \tilde{\mathbf{x}}_2 \quad \cdots \quad \tilde{\mathbf{x}}_N] \begin{bmatrix} \tilde{\mathbf{x}}_1^T \\ \tilde{\mathbf{x}}_2^T \\ \vdots \\ \tilde{\mathbf{x}}_N^T \end{bmatrix} \mathbf{v} \\ &= \mathbf{v}^T \mathbf{Q} \mathbf{v} \end{aligned}$$



Maximizing Directional Variance

- ❑ From previous slide: Directional variance $s_Z^2 = \frac{1}{N} \sum_{i=1}^N (z_i - \bar{z})^2 = \mathbf{v}^T \mathbf{Q} \mathbf{v}$
- ❑ Maximizing directional variance can be formulated as an optimization problem:
$$\max_{\mathbf{v}} \mathbf{v}^T \mathbf{Q} \mathbf{v} \quad \text{s.t.} \quad \|\mathbf{v}\| = 1$$
- ❑ Let $\mathbf{v}_1, \dots, \mathbf{v}_p$ be the **eigenvectors** of \mathbf{Q} : $\mathbf{Q} \mathbf{v}_j = \lambda_j \mathbf{v}_j$
- ❑ Sort eigenvalues in descending order: $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_p$
 - Can show that eigenvalues are real and non-negative
- ❑ **Theorem**: Any local maxima of the variance directional is an eigenvector
 - $\mathbf{v} = \mathbf{v}_j$ for some j and $\mathbf{v}^T \mathbf{Q} \mathbf{v} = \lambda_j$
 - Proof below



Visualizing Principal Components

□ **Principal components:** The eigenvectors of Q , v_1, \dots, v_p

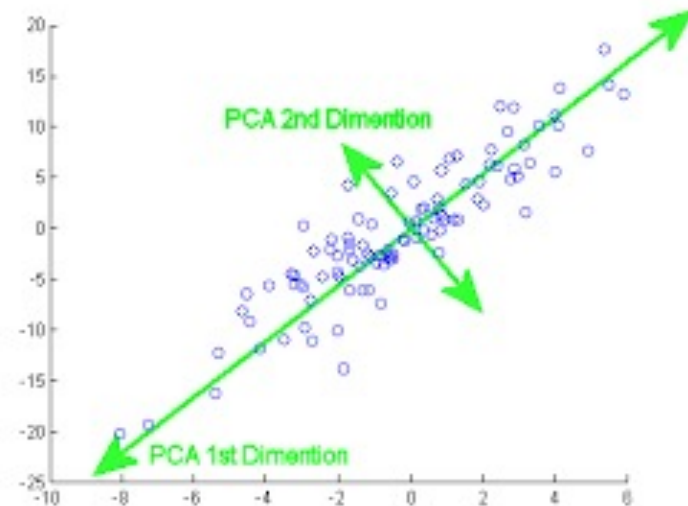
- Always normalized $\|v_j\| = 1$
- Sorted by eigenvalues $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_p$
- Each vector is of dimension p

□ **Key property:** Vectors are orthogonal

- $v_j^T v_k = 0$ if $j \neq k$

□ **Represents directions of decreasing variance:**

- v_1 : PC 1 = Direction of max variance
- v_2 : PC 2 = Direction of second most variance
- v_3 : PC 3 = Direction of third most variance
- ...



Proof PCs = Eigenvectors of Q

- PC constrained optimization problem:

$$\max_v \mathbf{v}^T \mathbf{Q} \mathbf{v} \quad \text{s.t.} \quad \|\mathbf{v}\| = 1$$

- Define **Lagrangian**: $L(\mathbf{v}, \lambda) = \mathbf{v}^T \mathbf{Q} \mathbf{v} - \lambda [\|\mathbf{v}\|^2 - 1]$

- At any local maxima:

$$\frac{\partial L}{\partial \mathbf{v}} = 0 \Rightarrow \mathbf{Q} \mathbf{v} - \lambda \mathbf{v} = \mathbf{0}$$

- This shows that \mathbf{v} is an eigenvector of Q



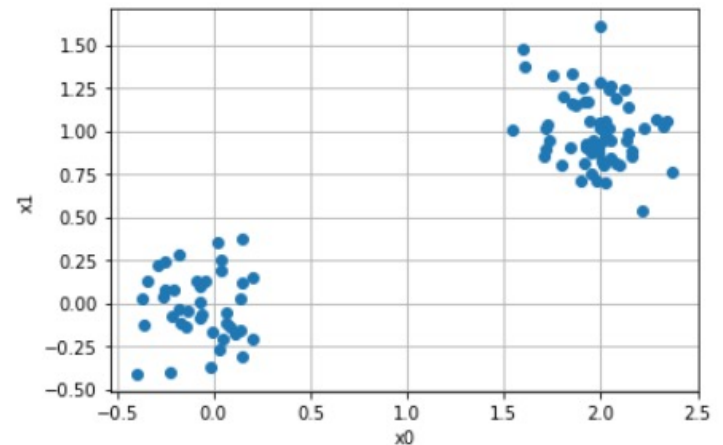
In-Class Exercise

Exercise 1


We begin by showing how to compute and visualize PCs manually on a simple 2-dim synthetic data set. First, run the following code to generate 100 samples of synthetic data. Each data point has `d=2` dimensions.

```
1 p = 0.5
2 std = 0.2
3 s = np.array([2,1])
4 d = 2
5 ns = 100
6
7 U = np.random.normal(0,std,(ns,d))
8 v = np.random.uniform(0,1,ns)
9 X = U + (v < p)[:None]*s[None,:]
10
```

Create a scatter plot of the data of the two features of the data, `X[:,0]` and `X[:,1]`.



Outline

- ☐ Dimensionality reduction
- ☐ Principal components and directions of variance
-  ☐ Approximation with PCs
- ☐ Computing PCs via the SVD
- ☐ Face example in python
- ☐ Training models from PCs
- ☐ Low rank approximations and recommender systems



Low-Dimensional Representations

□ Given data $\mathbf{x}_i, i = 1, \dots, N$. Each $\mathbf{x}_i \in \mathbb{R}^p$

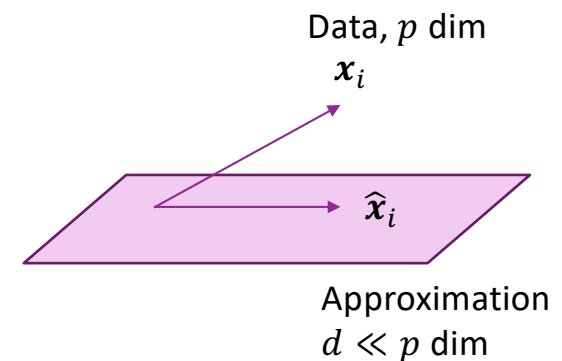
□ **Problem:** Find **basis vectors** $\mathbf{v}_j, j = 1, \dots, d$ such that:

$$\mathbf{x}_i \approx \hat{\mathbf{x}}_i = \bar{\mathbf{x}} + \sum_{j=1}^d \alpha_{ij} \mathbf{v}_j$$

- Sample mean + linear combination of basis vectors
- $\alpha_i = (\alpha_{i1}, \dots, \alpha_{id})$ is an approximate **coordinates** of \mathbf{x}_i in basis $(\mathbf{v}_1, \dots, \mathbf{v}_d)$

□ Dimensionality reduction:

- If $d \ll p$ we have represented \mathbf{v}_i with a smaller number of coefficients.



NYU

TANDON SCHOOL
OF ENGINEERING

18



Orthonormal Sets and Bases

□ **Definition:** A set of vectors $\mathbf{v}_1, \dots, \mathbf{v}_d$ are an **orthonormal set** if:

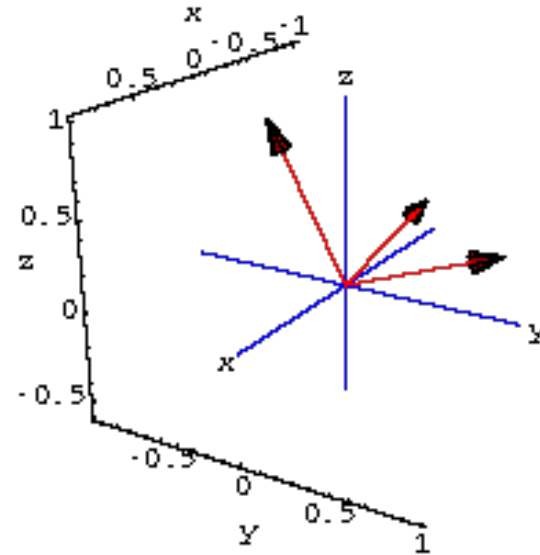
- $\|\mathbf{v}_j\| = 1$ for all j (unit length)
- $\mathbf{v}_j^T \mathbf{v}_k = 0$ if $j \neq k$ (perpendicular to one another)

□ **Matrix form:** If $\mathbf{V} = [\mathbf{v}_1 \dots \mathbf{v}_d]$, then $\mathbf{V}^T \mathbf{V} = \mathbf{I}_d$

□ If $d = p$ then $\mathbf{v}_1, \dots, \mathbf{v}_p$ is called an **orthonormal basis**

- \mathbf{V} is an **orthogonal matrix**

□ **Key property:** the PCs form an orthonormal basis



NYU

TANDON SCHOOL
OF ENGINEERING

Coefficients in an Orthonormal Basis

□ Suppose $\mathbf{v}_1, \dots, \mathbf{v}_p$ is an orthonormal basis

□ Given a vector \mathbf{z} , can write

$$\mathbf{z} = \sum_{j=1}^p \alpha_j \mathbf{v}_j, \quad \alpha_j = \mathbf{v}_j^T \mathbf{z}$$

- Simple expression for computing coefficients in an orthonormal basis

□ Matrix form:

$$\boldsymbol{\alpha} = \mathbf{V}^T \mathbf{z}, \quad \mathbf{z} = \mathbf{V} \boldsymbol{\alpha}$$



Approximating the Data Matrix

- Given data $\mathbf{x}_i, i = 1, \dots, N$
- Let $\mathbf{v}_1, \dots, \mathbf{v}_p$ be the PCs
- Find coefficient expansion of each data sample:

$$\mathbf{x}_i = \bar{\mathbf{x}} + \sum_{j=1}^p \alpha_{ij} \mathbf{v}_j, \quad \alpha_{ij} = \mathbf{v}_j^T (\mathbf{x}_i - \bar{\mathbf{x}})$$

- Approximation with d coefficients:

$$\hat{\mathbf{x}}_i = \bar{\mathbf{x}} + \sum_{j=1}^d \alpha_{ij} \mathbf{v}_j$$



Geometry of Approximations

□ Approximation can be interpreted geometrically

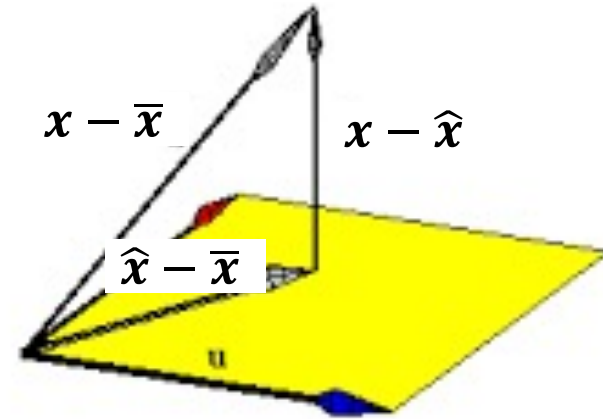
□ Let V be set of all linear combinations

$$\sum_{j=1}^d \alpha_j v_j$$

- V is a vector space
- Called the span of v_1, \dots, v_d

□ $\hat{x} - \bar{x}$ is the closest vector in V to $x - \bar{x}$

- Note the subtraction of the mean



Space spanned by v_1, \dots, v_d



NYU

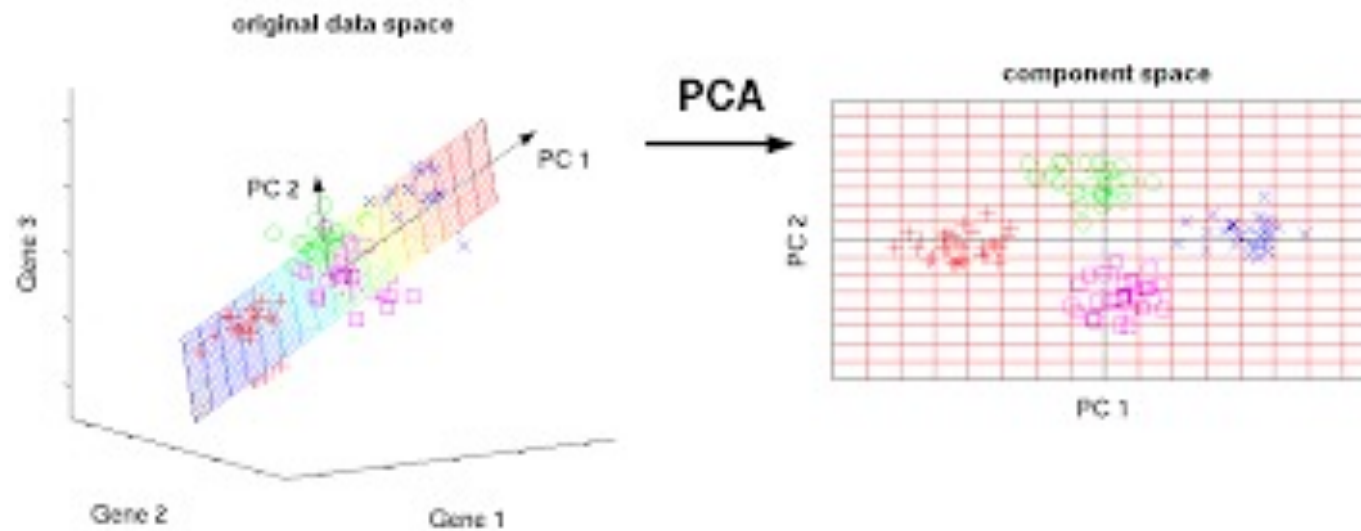
TANDON SCHOOL
OF ENGINEERING

22



Visualizing the Representation

- Finds a low-dimensional representation



NYU

TANDON SCHOOL
OF ENGINEERING

Example Calculation

□ Problem:

- Let $\mathbf{v}_1 = \frac{1}{\sqrt{2}} [1, 1, 0]$, $\mathbf{v}_2 = \frac{1}{\sqrt{6}} [1, -1, 2]$
- Show \mathbf{v}_1 and \mathbf{v}_2 are orthonormal

□ Solution:

- $\mathbf{v}_1^T \mathbf{v}_1 = \frac{1}{2} (1^2 + 1^2 + 0^2)$
- $\mathbf{v}_2^T \mathbf{v}_2 = \frac{1}{6} (1^2 + (-1)^2 + 2^2) = 1$
- $\mathbf{v}_1^T \mathbf{v}_2 = \frac{1}{\sqrt{2(3)}} (1(1) + 1(-1) + 0(2)) = 0$



Example Calculation Continued

□ Problem:

- Let $\mathbf{v}_1 = \frac{1}{\sqrt{2}} [1, 1, 0]$, $\mathbf{v}_2 = \frac{1}{\sqrt{6}} [1, -1, 2]$ be two PCs
- Let $\bar{\mathbf{x}} = [0, 1, 2]$ be the mean of the data
- Find the approximation of $\mathbf{x} = [2, 4, 4]$ with the two PCs

□ Solution:

- Subtract mean: $\mathbf{x} - \bar{\mathbf{x}} = [2, 3, 2]$
- Coeff on PC1: $\alpha_1 = \mathbf{v}_1^T (\mathbf{x} - \bar{\mathbf{x}}) = \frac{1}{\sqrt{2}} [2 + 3 + 0] = \frac{5}{\sqrt{2}}$
- Coeff on PC2: $\alpha_2 = \mathbf{v}_2^T (\mathbf{x} - \bar{\mathbf{x}}) = \frac{1}{\sqrt{6}} [2 - 3 + 4] = \frac{3}{\sqrt{6}}$
- Approximation: $\hat{\mathbf{x}} = \bar{\mathbf{x}} + \sum_{j=1}^d \alpha_j \mathbf{v}_j = [0, 1, 2] + \frac{5}{2} [1, 1, 0] + \frac{3}{6} [1, -1, 2] \approx [3, 3, 3]$



Average Approximation Error

- Change base of $\mathbf{x}_i - \bar{\mathbf{x}}$ to eigen vectors of sample variances:

$$\mathbf{x}_i - \bar{\mathbf{x}} = \sum_{j=1}^p \alpha_{ij} \mathbf{v}_j$$

Eigen vectors sorted by decreasing order of eigenvalues

- Let $\hat{\mathbf{x}}_i$ = approximation with d PCs

$$\hat{\mathbf{x}}_i - \bar{\mathbf{x}} = \sum_{j=1}^d \alpha_{ij} \mathbf{v}_j$$

- Error in sample i :

$$\mathbf{x}_i - \hat{\mathbf{x}}_i = \sum_{j=d+1}^p \alpha_{ij} \mathbf{v}_j$$

- Theorem:** Average error with a d PC approximation is:

$$\frac{1}{N} \sum_{i=1}^N \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2 = \sum_{j=d+1}^p \lambda_j$$

Sum of the smallest $p - d$ eigenvalues



Proportion of Variance (PoV)

- Total variance of data set: Trace of covariance matrix

$$\frac{1}{N} \sum_{i=1}^N \|x_i - \bar{x}\|^2 = \sum_{j=1}^p \lambda_j$$

- Average approximation error:

$$\frac{1}{N} \sum_{i=1}^N \|x_i - \hat{x}_i\|^2 = \sum_{j=d+1}^p \lambda_j$$

- The proportion of variance explained by d PCs is:

$$PoV(d) = \frac{\sum_{j=1}^d \lambda_j}{\sum_{j=1}^p \lambda_j}$$

- Measure of approximation error in using d PCs

Example

PC index	λ_i	POV(i)
1	10	10/14.3 \approx 0.70
2	4	14/14.3 \approx 0.98
3	0.2	14.2/14.3 \approx 0.99
4	0.1	14.3/14.3 = 1

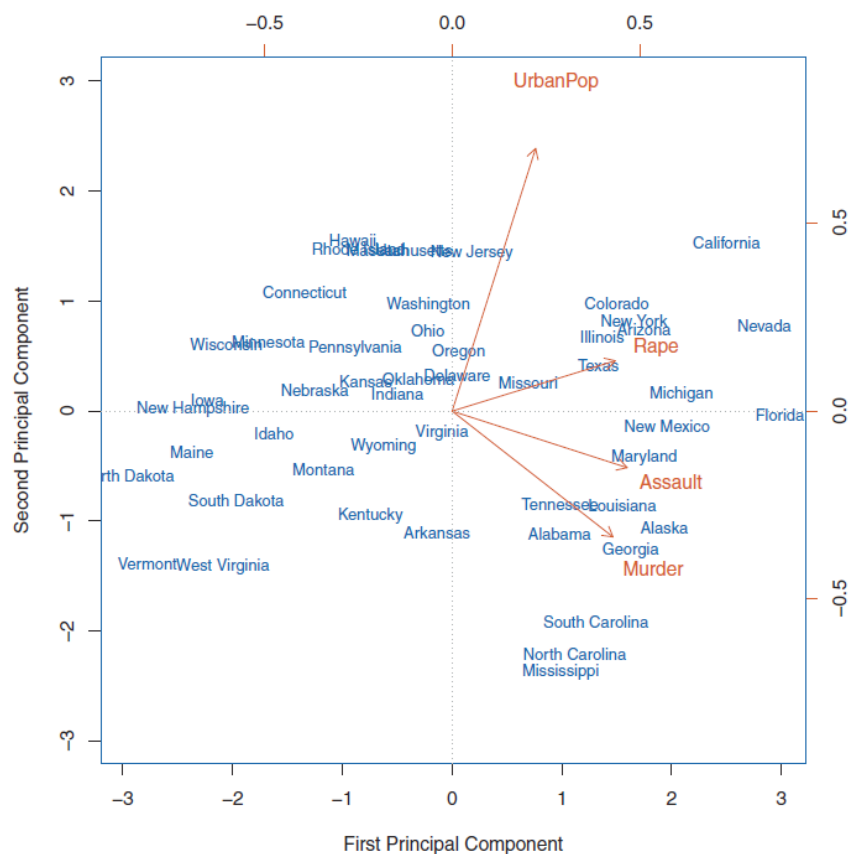


Latent Representations

- ❑ Each record is of the form: $\mathbf{x}_i \approx \bar{\mathbf{x}} + \sum_{j=1}^d \alpha_{ij} \mathbf{v}_j$
- ❑ Variance in \mathbf{x}_i explained by small number of “latent components”
 - Coefficients α_{ij} are the latent representations of \mathbf{x}_i
- ❑ Example:
 - \mathbf{x}_i = list of movie preferences for customer i
 - Movie preferences are highly correlated.
 - Could be explained by small number of components (action, romance, presence of stars, ...)
 - PCA can be used to find these out



Example: USArrests



- Arrests per capita in four categories
 - One record per US state
- Visualize PCA in a **biplot**
 - See the scores (i.e. coefficients of each state)
 - Loading (PC vectors)
- Fig from ISL 10.1



In-Class Exercise

Exercise 2: Computing the Approximation Error

We now verify the approximation errors. For each k :

- Compute \hat{X} the PC approximation of X using k PC coefficients
- Compute the approximation error, $err[k]$ the average error $\frac{1}{ns} \sum_{ij} (X[i,j] - \hat{X}[i,j])^2$
- Compute the expected approximation error, $err_pred[k]$ the expected approximation error based on the eigenvalues λ .

Remember you will need to sort the eigenvalues in descending order. You can use the command

```
I = np.argsort(lam)[::-1]
```




NYU

TANDON SCHOOL
OF ENGINEERING

30



Outline

- ☐ Dimensionality reduction
- ☐ Principal components and directions of variance
- ☐ Approximation with PCs
-  ☐ Computing PCs via the SVD
- ☐ Face example in python
- ☐ Training models from PCs
- ☐ Low rank approximations and recommender systems



NYU

TANDON SCHOOL
OF ENGINEERING

31



Singular Value Decomposition

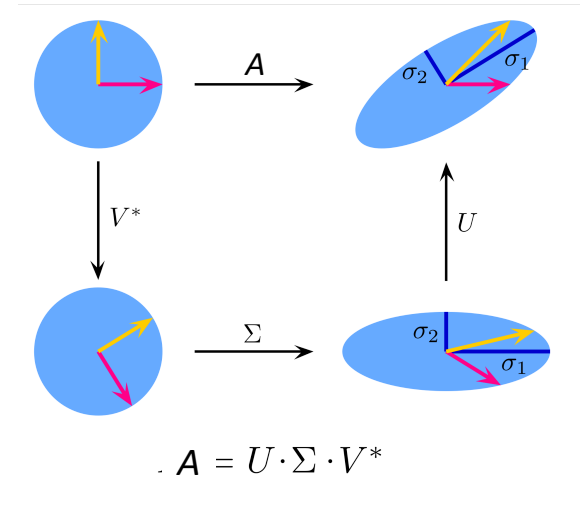
□ SVD: Powerful method in linear algebra

□ Given a matrix A :

- Decomposes the matrix into a product: $A = USV^T$
- Provides orthonormal bases of the input and output spaces
- Multiplication of A is equivalent to scaling in that basis

□ For PCA:

- Identifies low rank subspaces for data
- Computes coefficients in that subspace



Singular Value Decomposition Defined

- Given matrix $A \in \mathbb{F}^{n \times d}$
 - For PCA, this will be a scaled version of the data matrix

- SVD is $A = U\Sigma V^T$, where
 - $U \in \mathbb{F}^{n \times r}$, columns are orthonormal
 - $V \in \mathbb{F}^{d \times r}$, columns are orthonormal
 - $\Sigma = \text{diag}(s_1, \dots, s_r)$, sorted $s_1 \geq s_2 \geq \dots \geq s_r \geq 0$.
 - Called the singular values

- All matrices have an SVD
 - Matrices do not have to be square.

- Number of singular values $r \leq \min(n, d)$



Economy vs. Full SVD

□ Suppose $A \in \mathbb{R}^{n \times d}$ with rank $r \leq \min\{n, d\}$

□ Two types of SVDs

□ **Economy SVD:** $A = USV^*$

- $U \in \mathbb{F}^{n \times r}$, columns are orthonormal
- $V \in \mathbb{F}^{d \times r}$, columns are orthonormal
- $\Sigma \in \mathbb{F}^{r \times r}$ diagonal $\Sigma = \text{diag}(s_1, \dots, s_r)$,

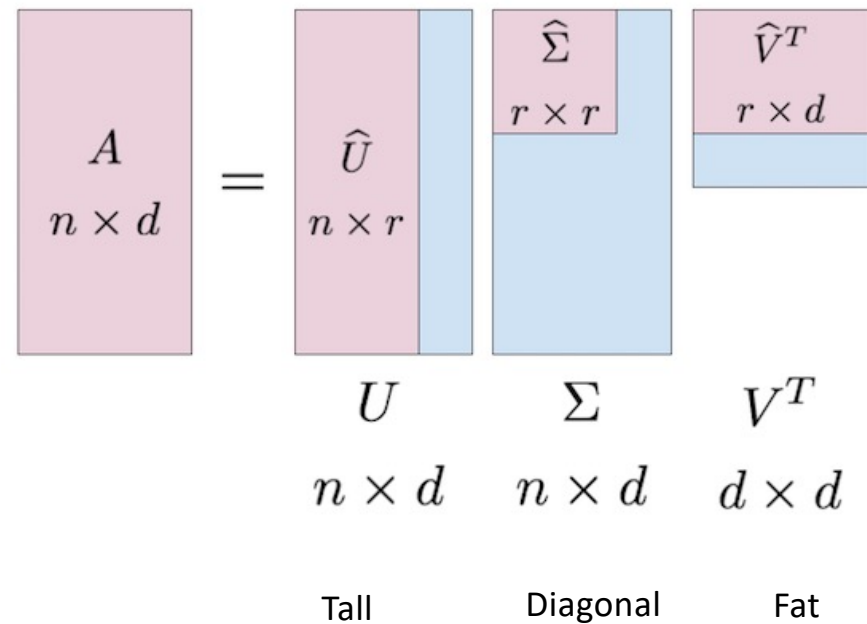
□ **Full SVD:** $A = USV^*$

- $U \in \mathbb{F}^{n \times n}$, columns are an orthonormal basis of \mathbb{R}^n
- $V \in \mathbb{F}^{d \times d}$, columns are an orthonormal basis of \mathbb{R}^d
- $\Sigma \in \mathbb{F}^{n \times d}$ with diagonal upper left $\Sigma = \begin{bmatrix} \hat{\Sigma} & 0 \\ 0 & 0 \end{bmatrix}$



SVD Visualized

- Pink matrices represent “economy” SVD
- Blue represent “full SVD”



Example

Let $A = \begin{bmatrix} 1 & 0 & 0 & 0 & 2 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \end{bmatrix}$

Then can check that $A = U\Sigma V^*$

$$U = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & \sqrt{5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad V^* = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \sqrt{0.2} & 0 & 0 & 0 & \sqrt{0.8} \\ 0 & 0 & 0 & 1 & 0 \\ -\sqrt{0.8} & 0 & 0 & 0 & \sqrt{0.2} \end{bmatrix}$$

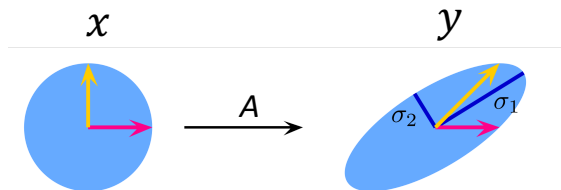
- Also verify that $UU^* = I_5$ and $VV^* = I_5$
- This can be found by (cleverly) permute the rows of A
- But, in general, use a computer to compute SVD



Geometric Interpretation

- Matrix can be viewed as a linear transformation/map

$$y = Ax$$



- Transform a vector x to a vector y

- $T: R^p \rightarrow R^n$
- The columns of A are the coordinates of the transformed basis vectors

- Check *Essence of linear algebra* on Youtube by 3Blue1Brown

- https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab



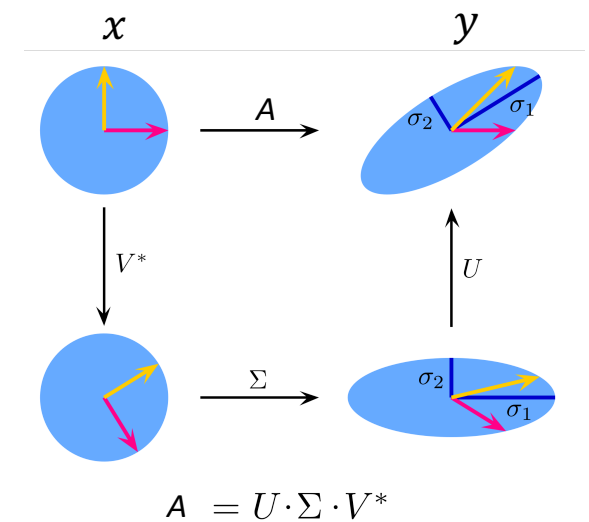
NYU

TANDON SCHOOL
OF ENGINEERING



Geometric Interpretation

- Let $A = U\Sigma V^*$ and $y = Ax$
- Consider a transformed space
 - $\mathbf{w} = \mathbf{V}^* \mathbf{x} = [w_1, \dots, w_N]$ coefficients in input basis $V = [v_1, \dots, v_N]$
 - $\mathbf{z} = \mathbf{U}^* \mathbf{y} = [z_1, \dots, z_M]$: coefficients in output basis $U = [u_1, \dots, u_M]$
- Then: $\mathbf{z} = \Sigma \mathbf{w}$ so $z_i = \sigma_i w_i$
- Each input direction \mathbf{v}_i is mapped to $\sigma_i \mathbf{u}_i$
- Consequence:
 - SVD finds orthonormal bases U, V such that matrix A is a linear scaling in each basis vector



Example Problem

- Suppose that $A = U\Sigma V^* \in \mathbb{R}^{3 \times 4}$ with $\Sigma = \text{diag}(3, 0.2, 0, 0)$
- If $\mathbf{x} = 2\mathbf{v}_1 + 3\mathbf{v}_2 + 4\mathbf{v}_3 + 5\mathbf{v}_4$ find $\mathbf{y} = A\mathbf{x}$ in terms of basis $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3$
- Solution:
 - $A\mathbf{v}_i = \sigma_i \mathbf{u}_i$ for all i
 - Therefore,

$$\begin{aligned}\mathbf{y} = A\mathbf{x} &= 2A\mathbf{v}_1 + 3A\mathbf{v}_2 + 4A\mathbf{v}_3 + 5A\mathbf{v}_4 \\ &= 2(3)\mathbf{u}_1 + 3(0.2)\mathbf{u}_2 + 4(0)\mathbf{u}_3 \\ &= 6\mathbf{u}_1 + 0.6\mathbf{u}_2\end{aligned}$$

Computing the SVD in Python

❑ Random matrix

```
# Create some random matrix  
A = np.random.normal(0,1,(100,10))
```

❑ Full SVD

```
# Full SVD  
U,s,Vtr = np.linalg.svd(A)
```

```
A.shape = (100, 10)  
U.shape = (100, 100)  
s.shape = (10,)  
Vtr.shape = (10, 10)
```

❑ Economy SVD

```
# Economy SVD  
U,s,Vtr = np.linalg.svd(A, full_matrices=False)
```

```
U.shape = (100, 10)  
s.shape = (10,)  
Vtr.shape = (10, 10)
```

❑ Reconstruction:

```
# Recovers back A  
Ahat = (U*s[None,:]).dot(Vtr)
```



Computing the PCA via SVD

- Let $\mathbf{A} = \frac{1}{\sqrt{N}} \tilde{\mathbf{X}}$ = scaled data matrix with sample mean removed.
- Take SVD: $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$
- Properties:
 - Sample covariance matrix is $\mathbf{Q} = \frac{1}{N} \tilde{\mathbf{X}}^T \tilde{\mathbf{X}} = \mathbf{A}^T \mathbf{A} = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^T \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \mathbf{V}\mathbf{\Sigma}^2 \mathbf{V}^T$
 - Eigenvalues of \mathbf{Q} = squared singular values of \mathbf{A}
 - PCs are \mathbf{v}_j , columns of \mathbf{V}
 - Coefficients are $\mathbf{Z} = \tilde{\mathbf{X}}\mathbf{V} = \sqrt{N}\mathbf{A}\mathbf{V} = \sqrt{N}\mathbf{U}\mathbf{\Sigma}$
- Hence, SVD provides PCs, eigenvalues coefficients \mathbf{Z} in the PCA representation.


In-Class Exercise

Exercise 3: Computing the PCA via the SVD

- Compute the matrix `A = 1/np.sqrt(ns)*Xm`
- Compute the economy SVD of `A` with the `np.linalg.svd()` command. Use `full_matrices=False` option. Print the dimensions of the components of the SVD



Outline

- Dimensionality reduction
- Principal components and directions of variance
- Approximation with PCs
- Computing PCs via the SVD
-  Face recognition using PCA in python
- Training models from PCs
- Low rank approximations and recommender systems



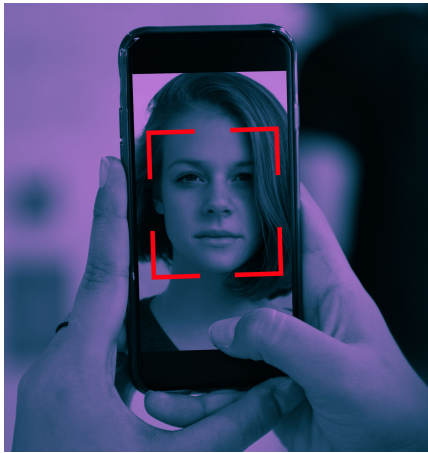
NYU

TANDON SCHOOL
OF ENGINEERING

43



Example: Face Recognition



Labeled Faces in the Wild Home



- ❑ Face recognition challenges:
 - Face images can be high-dimensional
 - We will use $50 \times 37 = 1850$ pixels
- ❑ Applying PCA:
 - Should be few degrees of freedom
 - Can transform to lower dimensional representations
- ❑ Data Labelled Faces in the Wild project
 - <http://vis-www.cs.umass.edu/lfw>
 - Large collection of faces (13000 images)
 - Taken from web articles about 20 years ago



Loading the Data

- ❑ Built-in routines to load data is scikit-learn
- ❑ Can take several minutes the first time (Be patient)

```
Image size      = 50 x 37 = 1850 pixels
Number faces    = 1288
Number classes  = 7
```

```
from sklearn.datasets import fetch_lfw_people
lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4)

2016-11-14 14:15:30,862 Downloading LFW metadata: http://vis-www.cs.umass.edu/lfw/pairsDevTrain.txt
2016-11-14 14:15:30,958 Downloading LFW metadata: http://vis-www.cs.umass.edu/lfw/pairsDevTest.txt
2016-11-14 14:15:31,028 Downloading LFW metadata: http://vis-www.cs.umass.edu/lfw/pairs.txt
2016-11-14 14:15:31,294 Downloading LFW data (~200MB): http://vis-www.cs.umass.edu/lfw/lfw-funneled.tgz
2016-11-14 14:20:10,056 Decompressing the data archive to C:\Users\Sundeeep\scikit_learn_data\lfw_home\lfw_funneled
2016-11-14 14:22:08,605 Loading LFW people faces from C:\Users\Sundeeep\scikit_learn_data\lfw_home
2016-11-14 14:22:09,735 Loading face #00001 / 01288
2016-11-14 14:22:13,640 Loading face #01001 / 01288
```



Plotting the Data

- ❑ Some example faces
- ❑ You may be too young to remember them all

Colin Powell



Colin Powell



Hugo Chavez



George W Bush



```
def plt_face(x):  
    h = 50  
    w = 37  
    plt.imshow(x.reshape((h, w)), cmap=plt.cm.gray)  
    plt.xticks([])  
    plt.yticks([])  
  
I = np.random.permutation(n_samples)  
plt.figure(figsize=(10,20))  
nplt = 4;  
for i in range(nplt):  
    ind = I[i]  
    plt.subplot(1,nplt,i+1)  
    plt_face(X[ind])  
    plt.title(target_names[y[ind]])
```



Computing the PCA

```
npix = h*w  
Xmean = np.mean(X,0)  
Xs = X - Xmean[None,:]
```

```
U,S,Vtr = np.linalg.svd(Xs, full_matrices=False)
```

❑ Manually compute the PCs with SVD

- Remove the mean
- Use broadcasting

◦ Compute the SVD

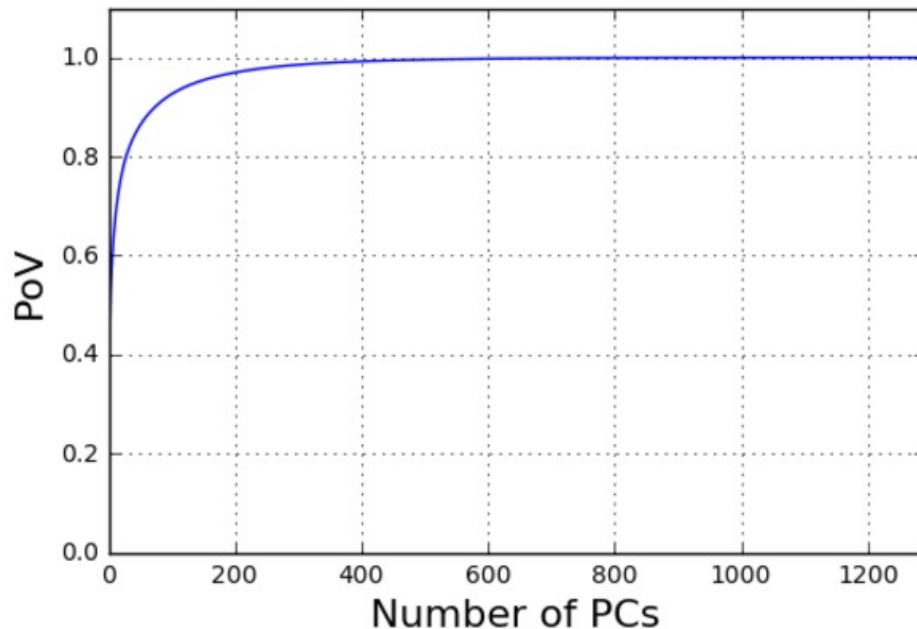
```
from sklearn.decomposition import PCA  
  
# Construct the PCA object  
pca = PCA(n_components=ncomp,  
          |svd_solver='randomized', whiten=True)  
  
# Fit the PCA components on the entire dataset  
pca.fit(X)
```

❑ Use sklearn builtin PCA function

- Construct a PCA object
- Call fit: Computes mean and PC components
- Stores values internally in the pca class



Finding the PoV



- ❑ Most variance explained in about 400 components
- ❑ Some reduction

```
lam = S**2
PoV = np.cumsum(lam)/np.sum(lam)

plt.plot(PoV)
plt.grid()
plt.axis([1,n_samples,0, 1.1])
plt.xlabel('Number of PCs', fontsize=16)
plt.ylabel('PoV', fontsize=16)
```



Plotting Approximations

```
nplt = 2          # number of faces to plot
ds = [0,5,10,20,100] # number of SVD approximations
use_pca = True    # True=Use sklearn reconstruction, else use SVD

# Loop over figures
iplt = 0
for ind in inds:
    for d in ds:
        plt.subplot(nplt,nd+1,iplt+1)
        if use_pca:
            # Zero out coefficients after d.
            # Note, we need to copy to not overwrite the coefficients
            Zd = np.copy(Z[ind,:])
            Zd[d:] = 0
            Xhati = pca.inverse_transform(Zd)
        else:
            # Reconstruct with SVD
            Xhati = (U[ind,:d]*S[None,:d]).dot(Vtr[:,d,:]) + Xmean

        plt_face(Xhati)
        plt.title('d={0:d}'.format(d))
        iplt += 1

# Plot the true face
plt.subplot(nplt,nd+1,iplt+1)
plt_face(X[ind,:])
plt.title('Full')
iplt += 1
```

□ Reconstruction using sklearn method

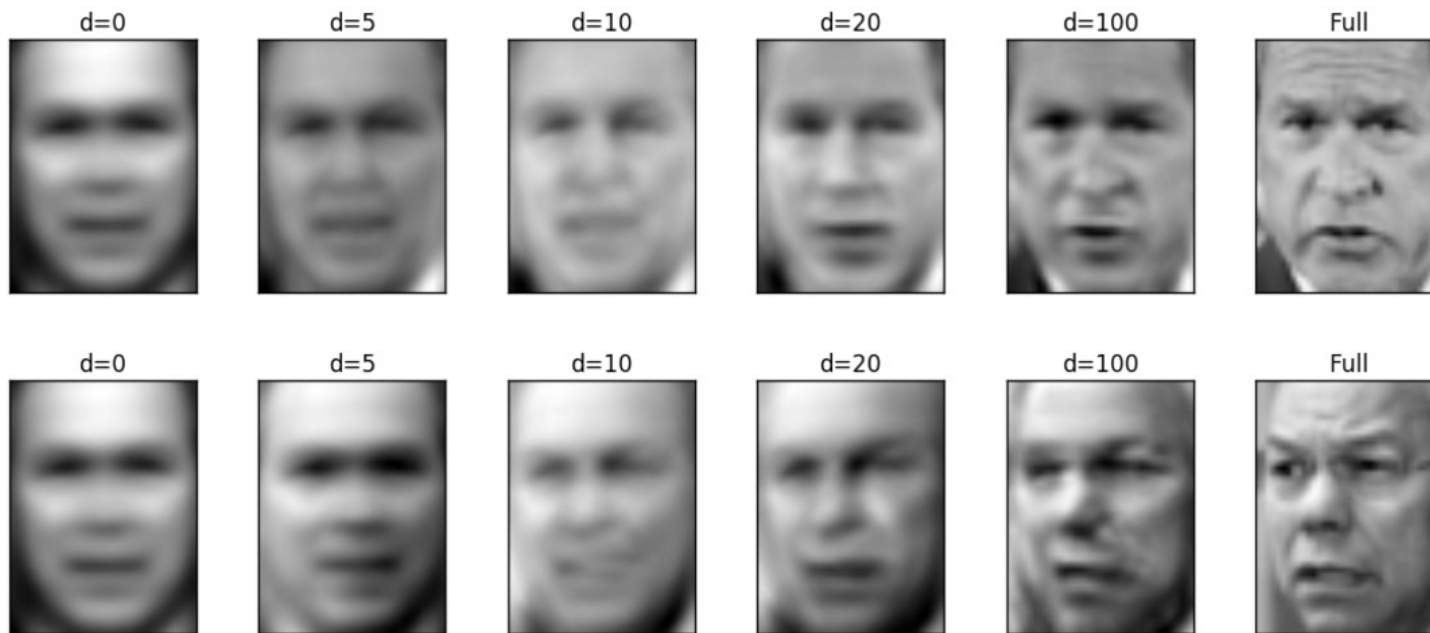
- Uses the inverse_transform method

□ Reconstruction using SVD

- Note use of broadcasting



Plotting the Approximations



NYU

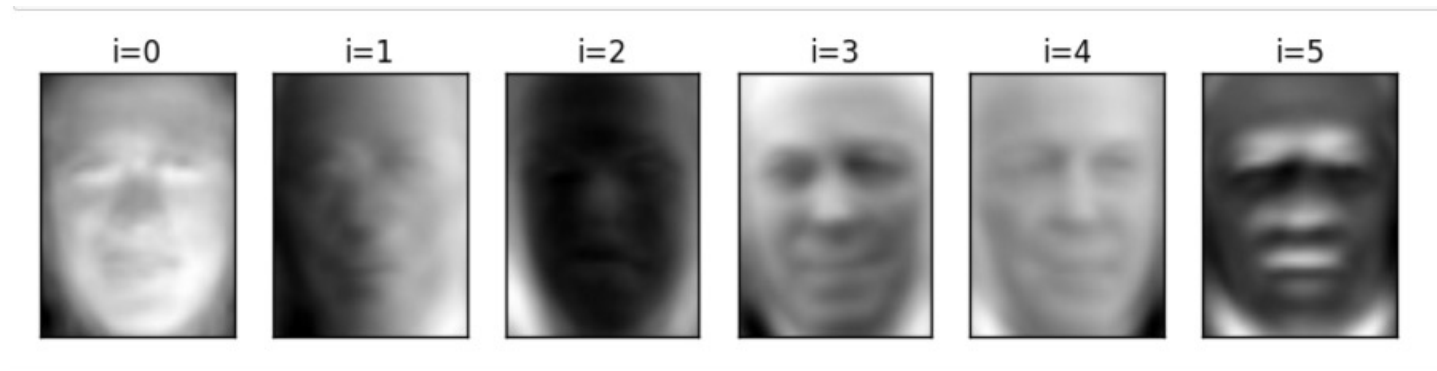
TANDON SCHOOL
OF ENGINEERING

50



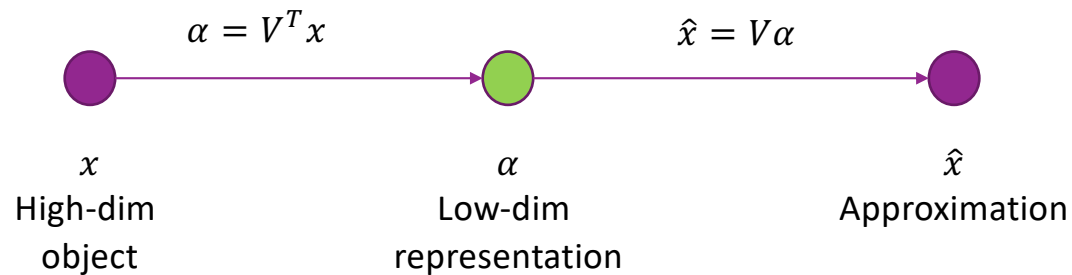
Plotting the PCs

- The PCs can be plotted as well



State-of-the-Art: Auto-Encoders

- ❑ PCA is a simple example of an **autoencoder**
- ❑ Tries to find low-dim representation
- ❑ Restricted to linear transforms
- ❑ Not very good for images and complex data



NYU

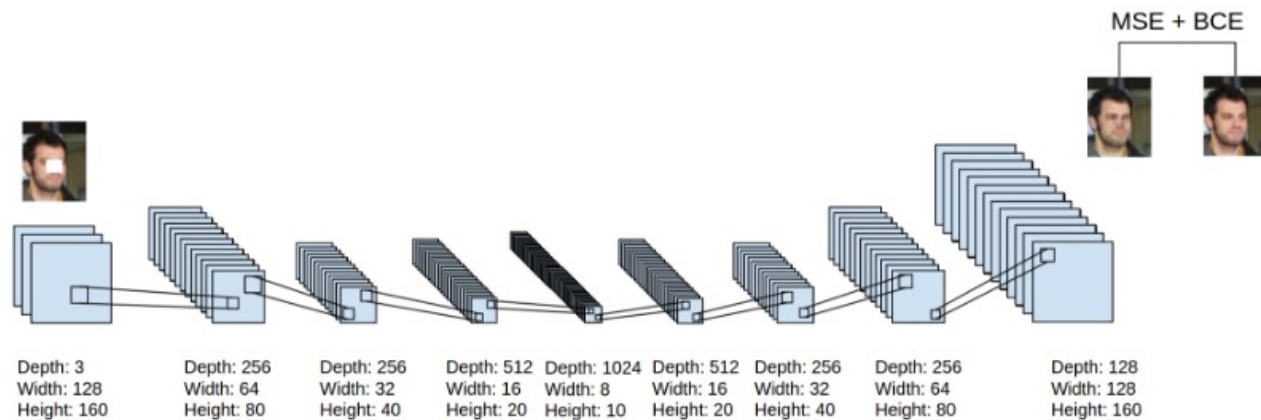
TANDON SCHOOL
OF ENGINEERING

52



Deep Auto-Encoders

- Can use deep networks for learning complex latent representations and their inverses
 - http://www.cc.gatech.edu/~hays/7476/projects/Avery_Wenchen/
 - <https://swarbrickjones.wordpress.com/2016/01/13/enhancing-images-using-deep-convolutional-generative-adversarial-networks-dcgans/> (Code in Theano not tensorflow)




NYU

TANDON SCHOOL
OF ENGINEERING

53



Outline

- ☐ Dimensionality reduction
- ☐ Principal components and directions of variance
- ☐ Approximation with PCs
- ☐ Computing PCs via the SVD
- ☐ Face example in python
-  ☐ Training models from PCs
- ☐ Low rank approximations and recommender systems



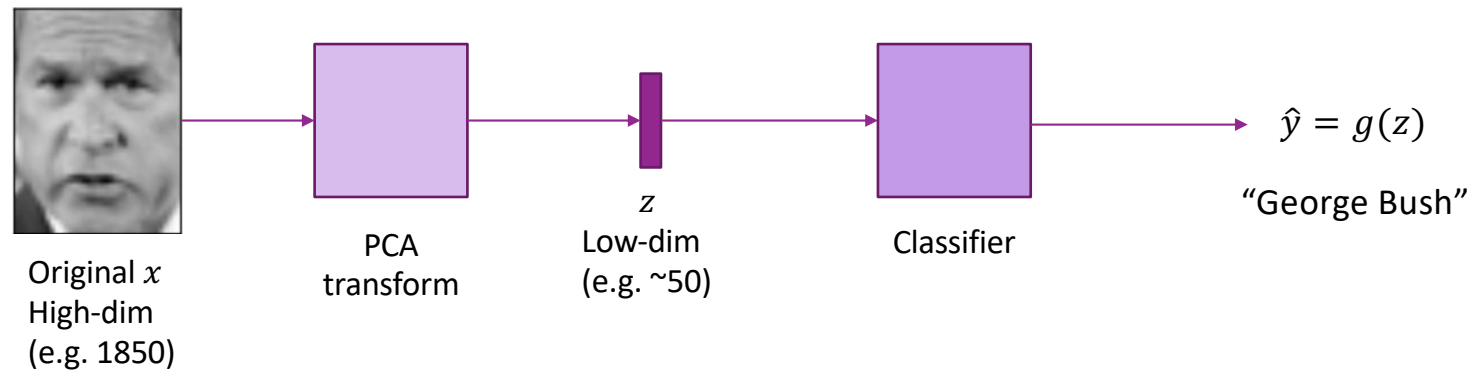
NYU

TANDON SCHOOL
OF ENGINEERING

54



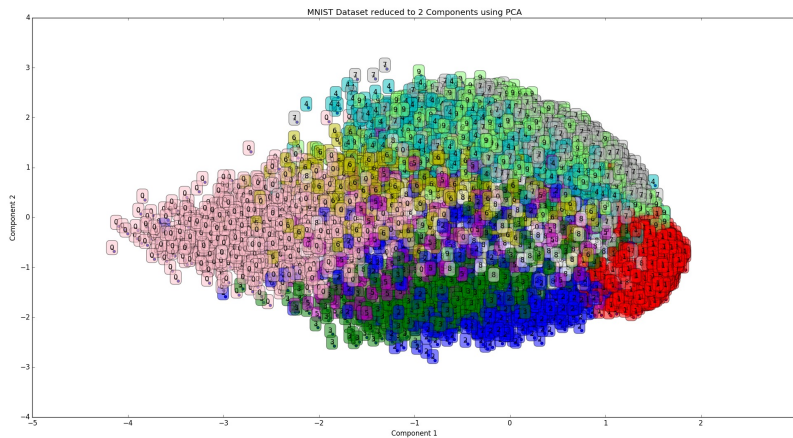
Classification Using PCs



- ❑ Many problems: Dimensionality of data x is too large
 - Classifier in original space will have too many parameters
- ❑ Key idea:
 - Learn a dimension reducing transform via PCA: $z = f(x)$
 - Train classifier on low-dim transform $\hat{y} = g(z)$



Why This Would Work?



□ PCA works if:
classes are separable in transformed domain

- Example to right:
- MNIST digits plotted in two PCs
 - Can mostly separate the classes



NYU

TANDON SCHOOL
OF ENGINEERING

56



Training and Testing

- ❑ Split data in training and test: $X_{tr}, y_{tr}, X_{ts}, y_{ts}$
- ❑ Fit PCA transform on $Z = g(X)$ on training data X_{tr}
 - Do not include test data in PCA fit!
 - Many students make this mistake.
- ❑ Transform training and test:
 - $Z_{tr} = g(X_{tr}), Z_{ts} = g(X_{ts})$
- ❑ Fit classifier $\hat{y} = f(z)$ on transformed training data (Z_{tr}, y_{tr})
- ❑ Predict classifier on transformed test data: $\hat{y}_{ts} = f(Z_{ts})$
- ❑ Score error rate / MSE on test data: $\epsilon = \frac{1}{N} \#\{\hat{y}_{ts}^i \neq y_{ts}^i\}$



Cross-Validation

- ❑ To find number of PCs and other parameters use cross-validation
- ❑ Split data in training and test: $X_{tr}, y_{tr}, X_{ts}, y_{ts}$
- ❑ For each set of parameters:
 - Fit PCA transform on $Z = g(X, \text{numPCs})$ on training data X_{tr}
 - Transform training and test: $Z_{tr} = g(X_{tr})$, $Z_{ts} = g(X_{ts})$
 - Fit classifier $\hat{y} = f(z)$ on transformed training data (Z_{tr}, y_{tr})
 - Predict classifier on transformed test data: $\hat{y}_{ts} = f(Z_{ts})$
 - Score (e.g. error rate / MSE) on test data: $\epsilon = \frac{1}{N} \#\{\hat{y}_{ts}^i \neq y_{ts}^i\}$
- ❑ Select the parameters with lowest score



Example: SVM classification with PCAs

```
npc_test = [25,50,75,100,200]
gam_test = [1e-3,4e-3,1e-2,1e-1]
C = 100
n0 = len(npc_test)
n1 = len(gam_test)
acc = np.zeros((n0,n1))
acc_max = 0

for i0, npc in enumerate(npc_test):

    # Fit PCA on the training data
    pca = PCA(n_components=npc, svd_solver='randomized', whiten=True)
    pca.fit(Xtr)

    # Transform the training and test
    Ztr = pca.transform(Xtr)
    Zts = pca.transform(Xts)

    for i1, gam in enumerate(gam_test):

        # Fitting on the transformed training data
        svc = SVC(C=C, kernel='rbf', gamma = gam)
        svc.fit(Ztr, ytr)

        # Predict on the test data
        yhat = svc.predict(Zts)

        # Compute the accuracy
        acc[i0,i1] = np.mean(yhat == yts)
        print('npc=%d gam=%12.4e acc=%12.4e' % (npc,gam,acc[i0,i1]))
```

❑ Parameters to search

- Number of PCs and gamma

❑ Fit on the training data.

- This is in the loop!

❑ Transform the data

❑ Fit classifier on transformed training data

❑ Test on the transformed test data

❑ Score on test data



NYU

TANDON SCHOOL
OF ENGINEERING

59



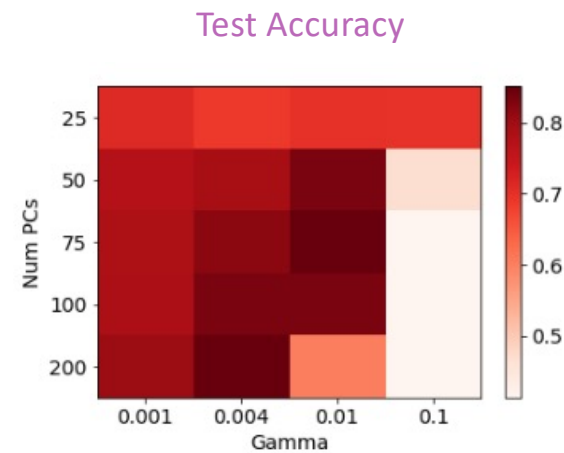
Example: Parameter Search

Search over:

- Number of PCs $\in \{25, 50, 75, 100, 200\}$
- $\gamma \in \{0.001, 0.004, 0.01, 0.1\}$

Plotted is the test accuracy

Best test accuracy $\approx 85\%$



Optimal num PCs = 75

Optimal gamma = 0.010000



Examples

Correct images

George W Bush George W Bush



George W Bush George W Bush



Original

Reduced

Error images

Tony Blair George W Bush



Gerhard Schroeder George W Bush



Original


Reduced



NYU

TANDON SCHOOL
OF ENGINEERING

Outline

- ❑ Dimensionality reduction
- ❑ Principal components and directions of variance
- ❑ Approximation with PCs
- ❑ Computing PCs via the SVD
- ❑ Face example in python
- ❑ Training models from PCs
-  Low rank approximations and recommender systems



NYU

TANDON SCHOOL
OF ENGINEERING

62



Low-Rank Approximations

- SVD can be used for a **low-rank approximation**
- SVD can be written: $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \sum_{j=1}^r \alpha_j \mathbf{u}_j \mathbf{v}_j^T$
- Consider k –term approximation: $\mathbf{X}_k = \sum_{j=1}^k \alpha_j \mathbf{u}_j \mathbf{v}_j^T$
- Properties:
 - \mathbf{X}_k is rank k
 - $\mathbf{X}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T$
 - Error is $\|\mathbf{X} - \mathbf{X}_k\|_F^2 = \sum_i \sum_j (X_{ij} - X_{k,ij})^2 = \sum_{j=k+1}^r \alpha_j^2$
 - If s_{k+1}, \dots, s_r is small then matrix is well approximated by rank k matrix

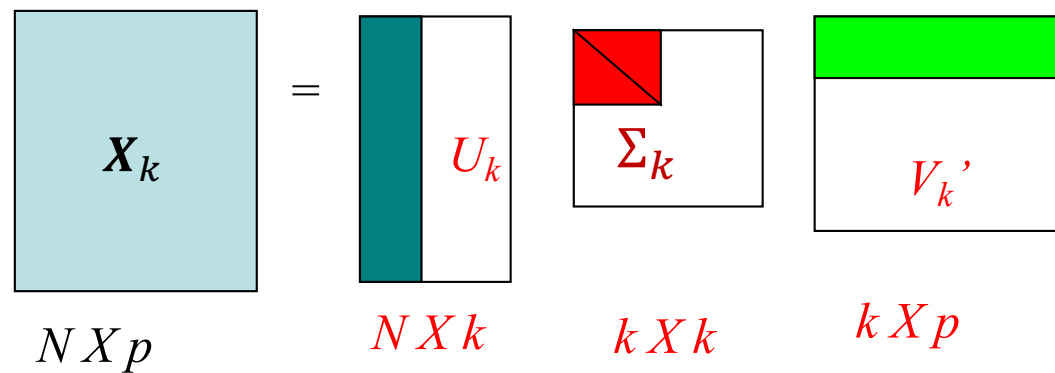


NYU

TANDON SCHOOL
OF ENGINEERING



Low-Rank Approximation Visualized



□ Can show: Reconstructed matrix X_k is optimal rank k approximation



NYU

TANDON SCHOOL
OF ENGINEERING



Recommender Systems

- ❑ How do you recommend a movie to a user?
- ❑ MovieLens dataset:
 - Get past ratings from users
 - Make recommendations for future

t[3]:

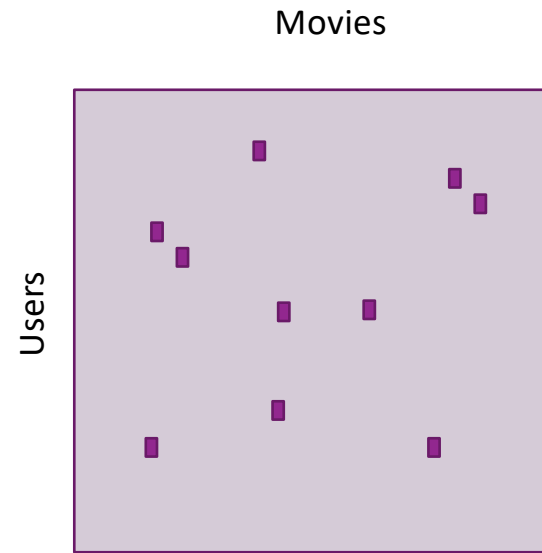
	movieId	title	genres
0	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy
1	2	Jumanji (1995)	Adventure Children Fantasy
2	3	Grumpier Old Men (1995)	Comedy Romance
3	4	Waiting to Exhale (1995)	Comedy Drama Romance
4	5	Father of the Bride Part II (1995)	Comedy



Ratings Matrix

- ❑ Data can be represented as **ratings matrix**
 - Users x movies
- ❑ **Problem:** Most users have only rated a small fraction
- ❑ Need to estimate unseen entries
 - Very sparse
- ❑ How can we do complete this matrix

Name	Dates	Users	Movies	Ratings	Density
ML Latest	'95 – '16	247,753	34,208	22,884,377	0.003%
ML Latest Small	'96 – '16	668	10,329	105,339	0.015%



Latent Factor Model for Ratings

□ Idea: Ratings for movies dependent on small number of **latent** factors

- E.g. Action, famous actors, genre, ...

□ Mathematically model as:

$$R_{ij} \approx \hat{R}_{ij} = b_i^u + b_j^m + \sum_{k=1}^K A_{ik} B_{jk}$$

- R_{ij} = Rating of movie j by user i
- b_i^u = Bias of user i
- b_j^m = Bias of movie j
- K = number of latent factors. Typically small $K \ll N_{user}, N_{movies}$
- A_{ik} = Preference of user i to factor k
- B_{jk} = Component of factor k in movie j



More to be added

- ❑ These slides are still under construction.
- ❑ More will be added on low rank approximations and embedding layers.



NYU

TANDON SCHOOL
OF ENGINEERING

68

