Unit 11
Principal Component Analysis

Prof. Phil Schniter

The Ohio State University

ECE 5307: Introduction to Machine Learning, Sp23

Learning objectives

- Recognize need for feature dimensionality reduction
- Understand PCA as RSS-minimizing linear approximation
  - Understand orthogonal projection
  - Recognize PCA as subspace fitting
  - Understand the role of the data-covariance eigenvectors in PCA
  - Know how to measure PCA performance using PoV
- Understand how to compute PCA using the SVD
- Understand how the PCA coefficients can be used in supervised learning tasks
- Understand how PCA can be used for data visualization
- Be familar with t-SNE and UNET, non-linear data-visualization techniques

Outline

- Dimensionality Reduction
- Principal Component Analysis (PCA)
- Computing PCA via the SVD
- Python Example: Eigenfaces and PCA-based Classification
- Data Visualization using PCA, t-SNE, and UMAP

Dimensionality reduction

- Many modern datasets have very high dimension d
- We would like to reduce the dimension (if possible) ...
  - to simplify classification/regression tasks
  - to save memory/storage space
  - to help visualize structure in data
- In this unit, we focus on dimensionality reduction via PCA
  - PCA is RSS-optimal *linear* dimensionality reduction
- We also briefly describe t-SNE and UMAP
  - Nonlinear dimensionality reduction techniques often used for visualization

Data representation

- Dataset: $\{\boldsymbol{x}_i\}_{i=1}^n$
  - Each sample has d features: $\boldsymbol{x}_i = [x_{i1}, \dots, x_{id}]^\mathsf{T} \in \mathbb{R}^d$
  - Can represent dataset using the matrix $\boldsymbol{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_n]^\mathsf{T} \in \mathbb{R}^{n \times d}$
  - Will assume data is centered (i.e., mean was removed, so $\sum_{i=1}^n \boldsymbol{x}_i = \mathbf{0}$)
- Note: there are *no targets/labels* here!
  - Either they don't exist, or we are ignoring them
  - This is known as unsupervised learning
  - Until now, we've focused on supervised learning, e.g., classification, regression
- What if data dimension d is very large?
  - Can we reduce the dimension to ease further data processing?

Example: Face data

- Face images can be high dimensional
  - We'll use the "Labeled Faces in the Wild (LFW)" dataset from 2007
  - These images contain $d = 50 \times 37 = 1850$ pixels
  - Modern face datasets are much larger, e.g., up to 1 million pixels
- As we will see, face images can be well approximated using a few coefficients
  - Can be "compressed"!
- How exactly do we do this?

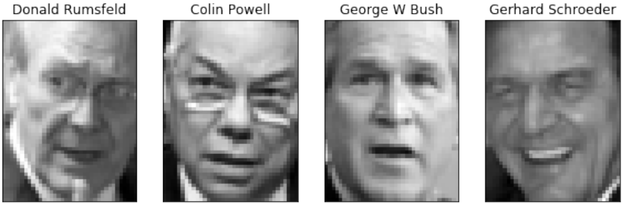
Loading the data

- The LFW face dataset is built into sklearn
  - The full collection contains $n = 13000$ images (from news stories in 2000s)
  - By requiring ≥ 70 faces per person, we extract a subset of 1288 images

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

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

- Some example faces:



Outline

- Dimensionality Reduction
- Principal Component Analysis (PCA)
- Computing PCA via the SVD
- Python Example: Eigenfaces and PCA-based Classification
- Data Visualization using PCA, t-SNE, and UMAP

PCA — Main ideas

- Main idea 1: Linearly approximate each feature vector $\boldsymbol{x}_i \in \mathbb{R}^d$ as follows:

$$\boldsymbol{x}_i \approx \boldsymbol{B}\boldsymbol{z}_i \text{ with } \boldsymbol{z}_i \in \mathbb{R}^R, \quad i = 1 \ldots n$$

  - The columns of $\boldsymbol{B} \in \mathbb{R}^{d \times R}$ form a "dictionary" with R elements
  - $\boldsymbol{z}_i$ contains R coefficients to linearly combine the dictionary elements for image i
  - $\boldsymbol{B}\boldsymbol{z}_i$ is a *linear* approximation of $\boldsymbol{x}_i$. (Linear is chosen for simplicity)
  - R is the "rank" of the approximation, where $1 \leq R < d$ (and ideally $R \ll d$)

- Main idea 2: Design the approximation to minimize RSS:

$$\left(\widehat{\boldsymbol{B}}, \{\widehat{\boldsymbol{z}}_i\}_{i=1}^n\right) = \arg\min_{\boldsymbol{B}, \{\boldsymbol{z}_i\}} \left\{ \sum_{i=1}^n \|\boldsymbol{x}_i - \boldsymbol{B}\boldsymbol{z}_i\|^2 \right\}$$

  - This is known as "principal component analysis" (PCA)
  - RSS is used for simplicity
    - Caveat: RSS may be poorly matched to downstream processing (e.g., classification)

PCA — Solution

- The optimal R-element approximation dictionary is

$$\boxed{\widehat{\boldsymbol{B}} = \boldsymbol{V}_R} \triangleq [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_R]$$

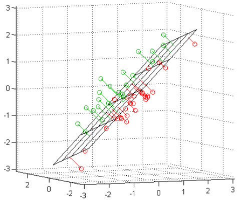
where $\{\boldsymbol{v}_r\}_{r=1}^R$ are the eigenvectors corresponding to the R largest eigenvalues $\{\lambda_r\}_{r=1}^R$ of the sample covariance matrix $\boldsymbol{Q}$:

$$\boldsymbol{Q} \triangleq \frac{1}{n}\sum_{i=1}^n \boldsymbol{x}_i\boldsymbol{x}_i^\mathsf{T} = \sum_{j=1}^d \lambda_j \boldsymbol{v}_j\boldsymbol{v}_j^\mathsf{T} \quad \text{since } \{\boldsymbol{x}_i\} \text{ is centered}$$

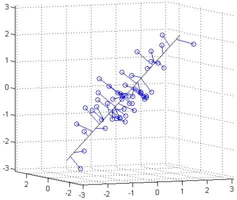
- The optimal coefficients for the approximation of $\boldsymbol{x}_i$ are $\boxed{\widehat{\boldsymbol{z}}_i = \boldsymbol{V}_R^\mathsf{T}\boldsymbol{x}_i}$

- The PCA approximation projects $\boldsymbol{x}_i \in \mathbb{R}^d$ onto the subspace spanned by $\{\boldsymbol{v}_r\}_{r=1}^R$, which are known as the R "principal components"

$d = 3, \; R = 2$



$d = 3, \; R = 1$



PCA — Derivation . . .

- Our derivation of PCA will proceed in two steps:
  1. Optimize the coefficients $\{\boldsymbol{z}_i\}$ for an arbitrary fixed dictionary $\boldsymbol{B}$
  2. Optimize the dictionary $\boldsymbol{B}$

- When optimizing $\boldsymbol{z}_i$, we encounter the familiar LS problem from Unit 2:

$$\widehat{\boldsymbol{z}}_i = \arg\min_{\boldsymbol{z}_i} \|\boldsymbol{x}_i - \boldsymbol{B}\boldsymbol{z}_i\|^2 = (\boldsymbol{B}^\mathsf{T}\boldsymbol{B})^{-1}\boldsymbol{B}^\mathsf{T}\boldsymbol{x}_i$$

- Plugging $\{\widehat{\boldsymbol{z}}_i\}_{i=1}^n$ back into the original problem yields

$$\begin{aligned}\widehat{\boldsymbol{B}} &= \arg\min_{\boldsymbol{B}} \left\{ \sum_{i=1}^n \left\|\boldsymbol{x}_i - \boldsymbol{B}(\boldsymbol{B}^\mathsf{T}\boldsymbol{B})^{-1}\boldsymbol{B}^\mathsf{T}\boldsymbol{x}_i\right\|^2 \right\} \\ &= \arg\min_{\boldsymbol{B}} \left\{ \sum_{i=1}^n \Big\| \underbrace{\left(\boldsymbol{I} - \boldsymbol{B}(\boldsymbol{B}^\mathsf{T}\boldsymbol{B})^{-1}\boldsymbol{B}^\mathsf{T}\right)}_{\triangleq\, \boldsymbol{P}_{\boldsymbol{B}}^{\perp}} \boldsymbol{x}_i \Big\|^2 \right\}\end{aligned}$$

- As we show next, we can interpret $\boldsymbol{P}_{\boldsymbol{B}}^{\perp}$ as an orthogonal projection matrix. . .

Orthogonal projection

- Consider the subspace $\mathrm{colsp}(\boldsymbol{B}) \triangleq \{\boldsymbol{B}\boldsymbol{z} \text{ s.t. } \boldsymbol{z} \in \mathbb{R}^R\} \subset \mathbb{R}^d$
  - the set of all linear combinations of the columns of $\boldsymbol{B}$
- The orthogonal projection of $\boldsymbol{x} \in \mathbb{R}^d$ onto $\mathrm{colsp}(\boldsymbol{B}) \ldots$
  - is the closest vector to $\boldsymbol{x}$ within $\mathrm{colsp}(\boldsymbol{B})$. So,

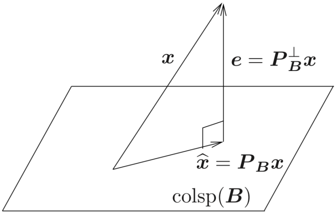
$$\boldsymbol{x} = \widehat{\boldsymbol{x}} + \boldsymbol{e}, \text{ where } \widehat{\boldsymbol{x}} \in \mathrm{colsp}(\boldsymbol{B}),\ \boldsymbol{e} \perp \mathrm{colsp}(\boldsymbol{B}),$$

  - which can be computed via

$$\widehat{\boldsymbol{x}} = \boldsymbol{P}_{\boldsymbol{B}}\boldsymbol{x} \text{ and } \boldsymbol{e} = \boldsymbol{P}_{\boldsymbol{B}}^{\perp}\boldsymbol{x}$$

  - using the orthogonal projection matrices

$$\boldsymbol{P}_{\boldsymbol{B}} \triangleq \boldsymbol{B}(\boldsymbol{B}^{\mathsf{T}}\boldsymbol{B})^{-1}\boldsymbol{B}^{\mathsf{T}} \text{ and } \boldsymbol{P}_{\boldsymbol{B}}^{\perp} \triangleq \boldsymbol{I} - \boldsymbol{P}_{\boldsymbol{B}}$$



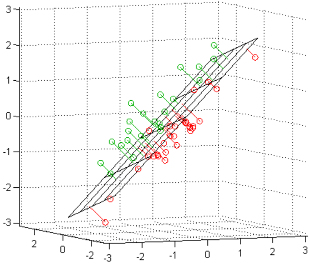
- Such matrices are symmetric and idempotent, i.e., $\boldsymbol{P}_{\boldsymbol{B}} = \boldsymbol{P}_{\boldsymbol{B}}^{\mathsf{T}}$ & $\boldsymbol{P}_{\boldsymbol{B}} = \boldsymbol{P}_{\boldsymbol{B}}^2$
- Note: $\boldsymbol{P}_{\boldsymbol{B}}$ has R eigenvalues $= 1$, and all other eigenvals $= 0$, while $\boldsymbol{P}_{\boldsymbol{B}}^{\perp}$ has R eigenvalues $= 0$, and all other eigenvals $= 1$

The role of projection in PCA

- Back to the PCA problem:

$$\widehat{\boldsymbol{B}} = \arg\min_{\boldsymbol{B}} \left\{ \sum_{i=1}^{n} \left\| \boldsymbol{P}_{\boldsymbol{B}}^{\perp} \boldsymbol{x}_i \right\|^2 \right\}$$

  - We now recognize $\boldsymbol{P}_{\boldsymbol{B}}^{\perp} \boldsymbol{x}_i$ as projection error
  - Thus, PCA chooses $\boldsymbol{B}$ to minimize the sum-squared projection error



- To further understand $\widehat{\boldsymbol{B}}$, we reformulate the above cost:

$$J(\boldsymbol{B}) \triangleq \sum_i \|\boldsymbol{P}_{\boldsymbol{B}}^{\perp} \boldsymbol{x}_i\|^2 = \sum_i \boldsymbol{x}_i^{\mathsf{T}} (\boldsymbol{P}_{\boldsymbol{B}}^{\perp})^{\mathsf{T}} \boldsymbol{P}_{\boldsymbol{B}}^{\perp} \boldsymbol{x}_i = \sum_i \boldsymbol{x}_i^{\mathsf{T}} \boldsymbol{P}_{\boldsymbol{B}}^{\perp} \boldsymbol{x}_i = \sum_i \operatorname{tr}\big(\boldsymbol{x}_i^{\mathsf{T}} \boldsymbol{P}_{\boldsymbol{B}}^{\perp} \boldsymbol{x}_i\big)$$

$$= \sum_i \operatorname{tr}\big(\boldsymbol{P}_{\boldsymbol{B}}^{\perp} \boldsymbol{x}_i \boldsymbol{x}_i^{\mathsf{T}}\big) = \operatorname{tr}\Big(\boldsymbol{P}_{\boldsymbol{B}}^{\perp} \sum_i \boldsymbol{x}_i \boldsymbol{x}_i^{\mathsf{T}}\Big) = n \operatorname{tr}\Bigg(\boldsymbol{P}_{\boldsymbol{B}}^{\perp} \underbrace{\frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i \boldsymbol{x}_i^{\mathsf{T}}}_{\text{sample covariance mtx } \boldsymbol{Q}}\Bigg)$$

where $\operatorname{tr}(\boldsymbol{A}) \triangleq \sum_j [\boldsymbol{A}]_{jj}$ and $\operatorname{tr}(\boldsymbol{AC}) = \operatorname{tr}(\boldsymbol{CA})$.

Eigen-decomposition of sample covariance matrices

- The sample covariance mtx $\boldsymbol{Q}$ is positive semi-definite, i.e., $\boldsymbol{x}^\mathsf{T}\boldsymbol{Q}\boldsymbol{x} \geq 0\ \forall \boldsymbol{x}$
  - Proof: $\boldsymbol{x}^\mathsf{T}\boldsymbol{Q}\boldsymbol{x} = \boldsymbol{x}^\mathsf{T}(\frac{1}{n}\sum_i \boldsymbol{x}_i\boldsymbol{x}_i^\mathsf{T})\boldsymbol{x} = \frac{1}{n}\sum_i(\boldsymbol{x}^\mathsf{T}\boldsymbol{x}_i)(\boldsymbol{x}_i^\mathsf{T}\boldsymbol{x}) = \frac{1}{n}\sum_i(\boldsymbol{x}^\mathsf{T}\boldsymbol{x}_i)^2 \geq 0$

- All positive semi-definite matrices have an eigen-decomposition of the form

$$\boldsymbol{Q} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\mathsf{T} \quad \text{where} \quad \begin{cases} \boldsymbol{V} \text{ is orthogonal (i.e., } \boldsymbol{V}\boldsymbol{V}^\mathsf{T} = \boldsymbol{I}_d = \boldsymbol{V}^\mathsf{T}\boldsymbol{V}) \\ \boldsymbol{\Lambda} = \mathrm{Diag}(\lambda_1, \ldots, \lambda_d) \text{ with } \lambda_j \geq 0\ \forall j \end{cases}$$

  Without loss of generality, we will assume $\{\lambda_j\}$ are sorted from large to small

Theorem (Eckart-Young, 1936)

The optimal $\boldsymbol{B} \in \mathbb{R}^{d\times R}$ is constructed from the R principal eigenvectors of $\boldsymbol{Q}$:

$$\widehat{\boldsymbol{B}} = \arg\min_{\boldsymbol{B}} n \operatorname{tr}\big(\boldsymbol{P}_{\boldsymbol{B}}^\perp \boldsymbol{Q}\big) = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_R] \triangleq \boldsymbol{V}_R,$$

More precisely, the optimal $\widehat{\boldsymbol{B}}$ is *any* $\boldsymbol{B}$ for which $\mathrm{colsp}(\boldsymbol{B}) = \mathrm{colsp}(\boldsymbol{V}_R)$

A simple proof of Eckart-Young

- Recall that we want to minimize the RSS cost

$$J(\boldsymbol{B}) = n\,\mathrm{tr}\big(\boldsymbol{P}_{\boldsymbol{B}}^{\perp}\boldsymbol{Q}\big) = n\,\mathrm{tr}\big((\boldsymbol{I}_d - \boldsymbol{P}_{\boldsymbol{B}})\boldsymbol{Q}\big) = n\,\mathrm{tr}(\boldsymbol{Q}) - n\,\mathrm{tr}(\boldsymbol{P}_{\boldsymbol{B}}\boldsymbol{Q})$$

- Equivalently, we can *maximize* the utility

$$U(\boldsymbol{B}) \triangleq \mathrm{tr}\big(\boldsymbol{P}_{\boldsymbol{B}}\boldsymbol{Q}\big) = \mathrm{tr}\big(\boldsymbol{P}_{\boldsymbol{B}}\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\mathsf{T}}\big) = \mathrm{tr}\big(\boldsymbol{V}^{\mathsf{T}}\boldsymbol{P}_{\boldsymbol{B}}\boldsymbol{V}\boldsymbol{\Lambda}\big) = \sum_{j=1}^{d}\alpha_j\lambda_j$$

$$\text{for} \quad \alpha_j \triangleq \big[\boldsymbol{V}^{\mathsf{T}}\boldsymbol{P}_{\boldsymbol{B}}\boldsymbol{V}\big]_{jj} = \boldsymbol{v}_j^{\mathsf{T}}\boldsymbol{P}_{\boldsymbol{B}}\boldsymbol{v}_j \in [0,1]$$

- Notice also that

$$\begin{aligned}\sum_{j=1}^{d}\alpha_j &= \mathrm{tr}\big(\boldsymbol{V}^{\mathsf{T}}\boldsymbol{P}_{\boldsymbol{B}}\boldsymbol{V}\big) = \mathrm{tr}\big(\boldsymbol{V}\boldsymbol{V}^{\mathsf{T}}\boldsymbol{P}_{\boldsymbol{B}}\big) = \mathrm{tr}\big(\boldsymbol{P}_{\boldsymbol{B}}\big) = \mathrm{tr}\big(\boldsymbol{B}(\boldsymbol{B}^{\mathsf{T}}\boldsymbol{B})^{-1}\boldsymbol{B}^{\mathsf{T}}\big)\\ &= \mathrm{tr}\big(\boldsymbol{B}^{\mathsf{T}}\boldsymbol{B}(\boldsymbol{B}^{\mathsf{T}}\boldsymbol{B})^{-1}\big) = \mathrm{tr}(\boldsymbol{I}_R) = R.\end{aligned}$$

- Thus we can consider the simplified optimization problem:

$$\text{Find } \{\alpha_j\}_{j=1}^{R} \text{ with } \alpha_j \in [0,1] \text{ and } \sum_{j=1}^{d}\alpha_j = R \text{ that maximizes } \sum_{j=1}^{d}\alpha_j\lambda_j$$

A simple proof of Eckart-Young (cont.)

- To intuitively solve the optimization problem ...

$$\text{Find } \{\alpha_j\}_{j=1}^R \text{ with } \alpha_j \in [0,1] \text{ and } \sum_{j=1}^d \alpha_j = R \text{ that maximizes } \sum_{j=1}^d \alpha_j \lambda_j$$

  - Think of α_j as a purchasing variable and λ_j as a reward for buying the jth item
  - You must buy between 0 and 1 units of each item, and R units total
  - Question: Which purchase is the most rewarding?
  - Answer: One unit each of the R best items! i.e., $\alpha_j = \begin{cases} 1 \text{ if } j = 1 \dots R \\ 0 \text{ if } j = R{+}1 \dots d \end{cases}$
  - Recall that $\{\lambda_j\}$ are ordered from large to small

- If $\boldsymbol{v}_j$ denotes the jth eigenvector of $\boldsymbol{Q}$, these optimal $\{\alpha_j\}$ are attained when

$$\boldsymbol{B} = [\boldsymbol{v}_1, \dots, \boldsymbol{v}_R] \triangleq \boldsymbol{V}_R, \quad \text{since} \quad \alpha_j = \boldsymbol{v}_j^\mathsf{T} \boldsymbol{P}_{\boldsymbol{B}} \boldsymbol{v}_j = \begin{cases} 1 \text{ if } j = 1 \dots R \\ 0 \text{ if } j = R{+}1 \dots d \end{cases}$$

$$\Rightarrow \boldsymbol{P}_{\boldsymbol{B}} = \boldsymbol{V}_R \boldsymbol{V}_R^\mathsf{T}$$

Summary of PCA

- Summary: Given centered data $\{\boldsymbol{x}_i\}_{i=1}^n$, PCA approximates $\boldsymbol{x}_i$ as

$$\widehat{\boldsymbol{x}}_i \approx \boldsymbol{V}_R \widehat{\boldsymbol{z}}_i \quad \text{with} \quad \widehat{\boldsymbol{z}}_i = \boldsymbol{V}_R^\mathsf{T} \boldsymbol{x}_i$$

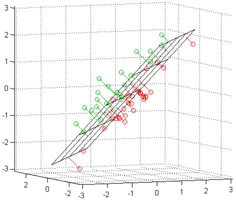
where $\boldsymbol{V}_R$ contains the R principal eigenvectors of the sample covariance mtx

$$\boldsymbol{Q} = \frac{1}{n}\sum_{i=1}^n \boldsymbol{x}_i \boldsymbol{x}_i^\mathsf{T} = \frac{1}{n}\boldsymbol{X}^\mathsf{T}\boldsymbol{X} \text{ with } \boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]^\mathsf{T} \in \mathbb{R}^{n \times d}$$

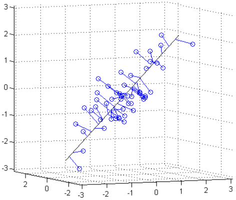
- These eigenvectors are called the "principal components"

- The PCA approximation projects $\boldsymbol{x}_i \in \mathbb{R}^d$ onto the subspace spanned by the R principal components of $\boldsymbol{Q}$

$d = 3, \ R = 2$



$d = 3, \ R = 1$



Performance of PCA

- How do we quantify the performance of PCA for a given rank R?
  - This will help in choosing R
- In scalar linear regression, we used $R^2 \triangleq 1 - \dfrac{\mathrm{RSS}}{ns_y^2}$
- For PCA, we will use proportion of variance, PoV $\triangleq 1 - \dfrac{\mathrm{RSS}}{n\,\mathrm{tr}(\boldsymbol{Q})}$
  - where $n\,\mathrm{tr}(\boldsymbol{Q}) = n\,\mathrm{tr}(\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\mathsf{T}}) = n\displaystyle\sum_{j=1}^{d}\lambda_j$
  - $\mathrm{RSS} = n\,\mathrm{tr}(\boldsymbol{P}_{\boldsymbol{B}}^{\perp}\boldsymbol{Q}) = n\,\mathrm{tr}(\boldsymbol{Q}) - n\,\mathrm{tr}(\boldsymbol{P}_{\boldsymbol{B}}\boldsymbol{Q}) = n\displaystyle\sum_{j=1}^{d}\lambda_j - n\sum_{j=1}^{R}\lambda_j = n\sum_{j=R+1}^{d}\lambda_j$
- Thus the PoV for R principal components is

$$\mathrm{PoV}(R) = \frac{n\sum_{j=1}^{d}\lambda_j}{n\sum_{j=1}^{d}\lambda_j} - \frac{n\sum_{j=R+1}^{d}\lambda_j}{n\sum_{j=1}^{d}\lambda_j} = \frac{\sum_{j=1}^{R}\lambda_j}{\sum_{j=1}^{d}\lambda_j}$$

. . . want close to 1

Outline

- Dimensionality Reduction
- Principal Component Analysis (PCA)
- Computing PCA via the SVD
- Python Example: Eigenfaces and PCA-based Classification
- Data Visualization using PCA, t-SNE, and UMAP

The singular value decomposition (SVD)

Given *any* matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, and denoting $r \triangleq \operatorname{rank}(\boldsymbol{X})$:

- The standard SVD decomposes $\boldsymbol{X}$ using square $\boldsymbol{U}$ & $\boldsymbol{V}$ as follows:

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top \quad \text{where} \quad \begin{cases} \boldsymbol{U} \in \mathbb{R}^{n \times n} & \text{obeys} \quad \boldsymbol{U}\boldsymbol{U}^\top = \boldsymbol{I}_n = \boldsymbol{U}^\top\boldsymbol{U} \\ \boldsymbol{V} \in \mathbb{R}^{d \times d} & \text{obeys} \quad \boldsymbol{V}\boldsymbol{V}^\top = \boldsymbol{I}_d = \boldsymbol{V}^\top\boldsymbol{V} \\ \boldsymbol{S} \in \mathbb{R}^{n \times d} & \text{obeys} \quad \boldsymbol{S} = \operatorname{Diag}(s_1, \dots, s_m) \\ & \qquad\quad \text{where} \quad m = \min\{n, d\} \\ & \qquad\quad \text{and} \quad s_1 \geq s_2 \geq s_3 \geq \cdots \geq 0 \end{cases}$$

- The "economy SVD" decomposes $\boldsymbol{X}$ using square $\boldsymbol{S}_r$ and tall $\boldsymbol{U}_r$ & $\boldsymbol{V}_r$:

$$\boldsymbol{X} = \boldsymbol{U}_r\boldsymbol{S}_r\boldsymbol{V}_r^\top \quad \text{where} \quad \begin{cases} \boldsymbol{U}_r \in \mathbb{R}^{n \times r} & \text{obeys} \quad \boldsymbol{U}_r^\top\boldsymbol{U}_r = \boldsymbol{I}_r \\ \boldsymbol{V}_r \in \mathbb{R}^{d \times r} & \text{obeys} \quad \boldsymbol{V}_r^\top\boldsymbol{V}_r = \boldsymbol{I}_r \\ \boldsymbol{S}_r \in \mathbb{R}^{r \times r} & \text{obeys} \quad \boldsymbol{S}_r = \operatorname{Diag}(s_1, \dots, s_r) \\ & \qquad\quad \text{where} \quad r \leq \min\{n, d\} \\ & \qquad\quad \text{and} \quad s_1 \geq s_2 \geq \cdots \geq s_r > 0 \end{cases}$$

Computing PCA via the standard SVD

- As before, assume that $\boldsymbol{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_n]^\mathsf{T} \in \mathbb{R}^{n \times d}$ is centered
  - That is, the sample mean of every column equals zero
- For PCA, we used the (sorted) eigen-decomposition of the covariance matrix

$$\boldsymbol{Q} = \frac{1}{n}\boldsymbol{X}^\mathsf{T}\boldsymbol{X} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\mathsf{T} \;\text{ where } \begin{cases} \boldsymbol{V}\boldsymbol{V}^\mathsf{T} = \boldsymbol{I}_d = \boldsymbol{V}^\mathsf{T}\boldsymbol{V} \\ \boldsymbol{\Lambda} = \mathrm{Diag}(\lambda_1, \dots, \lambda_d) \\ \lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \cdots \geq 0 \end{cases}$$

- Plugging in the standard SVD of $\boldsymbol{X}$, i.e., $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\mathsf{T}$, we find

$$\boldsymbol{Q} = \frac{1}{n}\boldsymbol{V}\boldsymbol{S}^\mathsf{T}\underbrace{\boldsymbol{U}^\mathsf{T}\boldsymbol{U}}_{\boldsymbol{I}_n}\boldsymbol{S}\boldsymbol{V}^\mathsf{T} = \boldsymbol{V}(\tfrac{1}{n}\boldsymbol{S}^\mathsf{T}\boldsymbol{S})\boldsymbol{V}^\mathsf{T} \text{ where } \begin{cases} \boldsymbol{V}\boldsymbol{V}^\mathsf{T} = \boldsymbol{I}_d = \boldsymbol{V}^\mathsf{T}\boldsymbol{V} \\ \frac{1}{n}\boldsymbol{S}^\mathsf{T}\boldsymbol{S} = \mathrm{Diag}(\frac{s_1^2}{n}, \dots, \frac{s_d^2}{n}) \\ \frac{s_1^2}{n} \geq \frac{s_2^2}{n} \geq \frac{s_3^2}{n} \geq \cdots \geq 0 \end{cases}$$

- So, the quantities $\boldsymbol{V}, \{s_j\}$ from the SVD of $\boldsymbol{X}$ are sufficient to compute PCA:
  - $\lambda_j = s_j^2/n$ and the $\boldsymbol{V}$ matrices are the same (if eigenvals are distinct and sorted)

Computing PCA via the economy SVD

- Remember: the economy SVD computes only the top r singular vectors, i.e., $\boldsymbol{U}_r$ and $\boldsymbol{V}_r$, where $r = \operatorname{rank}(\boldsymbol{X}) \leq \min(n, d)$
  - Thus, it may require less computational complexity than the standard SVD
- For PCA, need to compute only the top R singular vectors $\boldsymbol{V}_R$, where $R < r$
  - R is a design choice, and typically $R \ll r$
- Thus it's more efficient to use the economy SVD when computing PCA:

$$\begin{aligned}(\boldsymbol{U}_r, \boldsymbol{S}_r, \boldsymbol{V}_r^{\mathsf{T}}) &= \mathsf{SVD}_{\mathsf{economy}}(\boldsymbol{X}) \\ \boldsymbol{V}_R &= \text{first } R \text{ columns of } \boldsymbol{V}_r \\ \boldsymbol{z}_i &= \boldsymbol{V}_R^{\mathsf{T}} \boldsymbol{x}_i \ \forall i = 1 \dots n\end{aligned}$$

Standardizing the PCA coefficients

- After computing the PCA coefficients $\{\boldsymbol{z}_i\}$, we might use them for classification or regression (assuming that we also have some targets $\{y_i\}$)
- If so, we should standardize our new features $\{\boldsymbol{z}_i\}$. For this, we want
  - $\overline{\boldsymbol{z}} \triangleq \frac{1}{n}\sum_{i=1}^n \boldsymbol{z}_i = \boldsymbol{0}$
  - $\frac{1}{n}\sum_{i=1}^n z_{ij}^2 = 1$ for all $j = 1 \ldots R \quad \Leftrightarrow \quad \mathrm{Diag}\Big(\overbrace{\frac{1}{n}\sum_{i=1}^n \boldsymbol{z}_i\boldsymbol{z}_i^{\mathsf{T}}}^{\triangleq \boldsymbol{Q}_z}\Big) = \boldsymbol{1}$
- Let's analyze the actual values of $\overline{\boldsymbol{z}}$ and $\mathrm{Diag}(\boldsymbol{Q}_z)$:
  - $\overline{\boldsymbol{z}} = \frac{1}{n}\sum_{i=1}^n \boldsymbol{V}_R^{\mathsf{T}}\boldsymbol{x}_i = \boldsymbol{V}_R^{\mathsf{T}}(\frac{1}{n}\sum_{i=1}^n \boldsymbol{x}_i) = \boldsymbol{V}_R^{\mathsf{T}}\boldsymbol{0} = \boldsymbol{0}$, since $\{\boldsymbol{x}_i\}$ are centered
  - $\boldsymbol{Q}_z = \frac{1}{n}\sum_{i=1}^n \boldsymbol{z}_i\boldsymbol{z}_i^{\mathsf{T}} = \boldsymbol{V}_R^{\mathsf{T}}(\frac{1}{n}\sum_{i=1}^n \boldsymbol{x}_i\boldsymbol{x}_i^{\mathsf{T}})\boldsymbol{V}_R = \boldsymbol{V}_R^{\mathsf{T}}\boldsymbol{Q}\boldsymbol{V}_R = \frac{1}{n}\boldsymbol{V}_R^{\mathsf{T}}\boldsymbol{V}_r\boldsymbol{S}_r^2\boldsymbol{V}_r^{\mathsf{T}}\boldsymbol{V}_R = \mathrm{Diag}(\frac{s_1^2}{n}, \ldots, \frac{s_R^2}{n})$ since $\boldsymbol{V}_R^{\mathsf{T}}\boldsymbol{V}_r = [\boldsymbol{I}_R \ \boldsymbol{0}_{R\times(r-R)}]$
- This implies that a standardized version of $\{\boldsymbol{z}_i\}$ is given by $\{\widetilde{\boldsymbol{z}}_i\}$ with
  - $\widetilde{\boldsymbol{z}}_i \triangleq \mathrm{Diag}\left(\frac{\sqrt{n}}{s_1}, \ldots, \frac{\sqrt{n}}{s_R}\right)\boldsymbol{z}_i$

Outline

PCA on the LFW face dataset

- First we center the data $\{\boldsymbol{x}_i\}$

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

- Then we compute the economy SVD

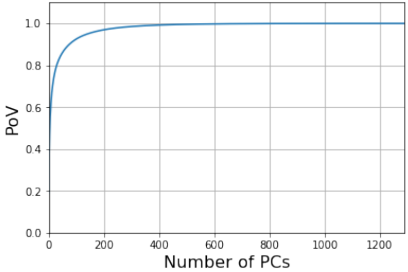
```
U,S,VT = np.linalg.svd(Xs, full_matrices=False)
VT.shape
```

```
(1288, 1850)
```

- Note that $\boldsymbol{V}_r^\mathsf{T}$ is wide, as we expect
- Then we compute the eigenvalues $\{\lambda_j\}_{j=1}^r$

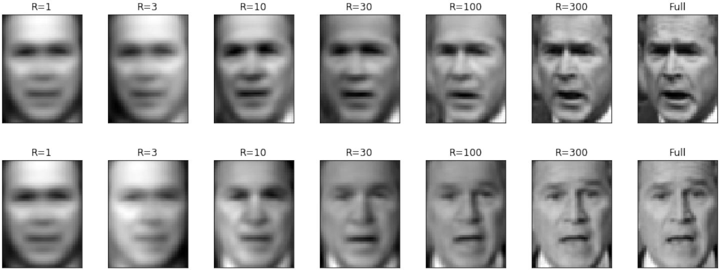
```
lam = S**2 / n_samples
PoV = np.cumsum(lam)/np.sum(lam)
```

- And finally we compute the Proportion-of-Variance
- The PoV plot suggests that $R = 400$ principal components capture nearly all the variance of our data

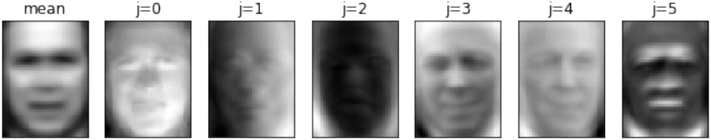


PCA approximation and eigenfaces

- We now show the PCA approximations versus R for two faces:



- And the mean & top 5 principal components $\{\boldsymbol{v}_j\}$ (i.e., "eigenfaces"):



Face recognition using the PCA coefficients

We now demonstrate classification (i.e., face recognition) via PCA coefficients

- Split data into training and test:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, stratify = y, random_state=43)
```

- Center the training:
- Perform economy SVD:

```
n_samples, _ = X_train.shape
Xtr_mean = np.mean(X_train,0)
Xtr = X_train - Xtr_mean[None,:]
Utr,Str,VTtr = np.linalg.svd(Xtr, full_matrices=False)
```

- Choose $R = 100$:
- Compute PCA coefficients $\boldsymbol{z}_i^\mathsf{T} = \boldsymbol{x}_i^\mathsf{T}\boldsymbol{V}_R\ \forall i$:

```
npc = 100
eigenfaces = VTtr[:npc,:]
Ztr = Xtr.dot(eigenfaces.T)
```

- Standardize the PCA coefficients: $\widetilde{\boldsymbol{z}}_i = \mathrm{Diag}\left(\frac{\sqrt{n}}{s_1}, \ldots, \frac{\sqrt{n}}{s_R}\right)\boldsymbol{z}_i$:

```
Ztr_s = Ztr / Str[None,:npc] * np.sqrt(n_samples)
```

Face recognition using the PCA coefficients (cont.)

- Tune an SVM classifier over regularization C & RBF kernel width γ:

```
param_grid = {'C': [1, 3, 10, 30, 100, 300],
              'gamma': [0.00001, 0.00003, 0.0001, 0.0003, 0.001, 0.003, 0.01], }
clf = GridSearchCV(SVC(kernel='rbf', class_weight='balanced'), param_grid, cv=5, iid=False)
clf = clf.fit(Ztr_s, y_train)
print("Best estimator found by grid search:")
print(clf.best_estimator_)
print("Cross validation accuracy with best estimator:")
print(clf.best_score_)
```

```
Best estimator found by grid search:
SVC(C=3, cache_size=200, class_weight='balanced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.003, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
Cross validation accuracy with best estimator:
0.8506081737123565
```

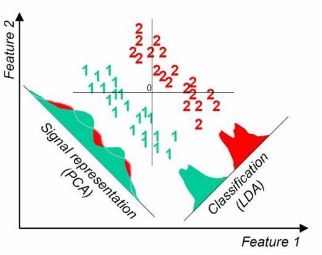
- After pre-processing the test data in the same way as training, classify it:

```
Xts = X_test - Xtr_mean[None,:]
Zts = Xts.dot(eigenfaces.T)
Zts_s = Zts / Str[None,:npc] * np.sqrt(n_samples)
y_hat = clf.predict(Zts_s)
acc = np.mean(y_hat==y_test)
print("The model accuracy on the test set is %f" % acc)
```

```
The model accuracy on the test set is 0.829193
```

Limitations of PCA

- As a dimensionality reduction technique, PCA has two main characteristics: 1) it is linear, and 2) it minimizes RSS.
  - These characteristics are convenient and allow us to derive a closed-form solution for PCA
- But minimizing RSS is not well justified for tasks like classification
  - For example, PCA can destroy the linear separability of a dataset, as illustrated here:

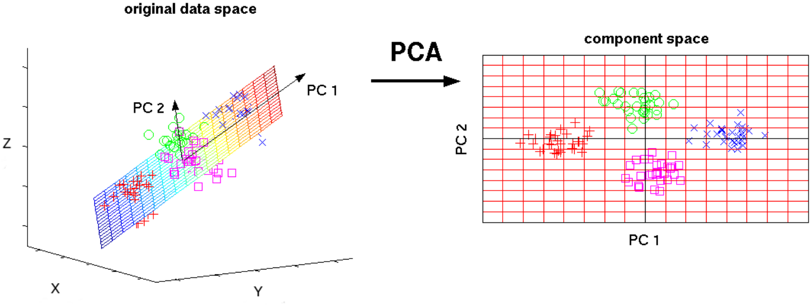


- Linear discriminant analysis (LDA) is a different form of linear dimensionality reduction that explicitly aims to discriminate between two classes
  - Although it's a nice idea, LDA isn't widely used today

Outline

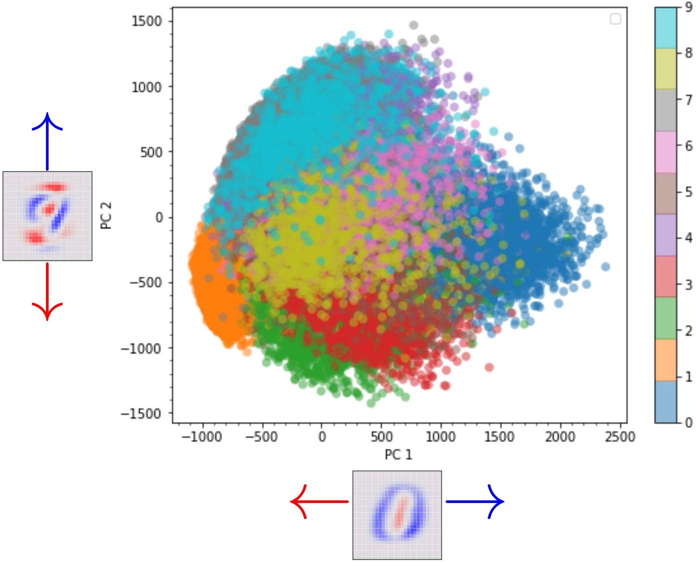
Data visualization using PCA

- When $d \geq 3$, the scatter plot of $\{\boldsymbol{x}_i\}_{i=1}^n$ can be difficult to visualize
- But when $R \leq 2$, the scatter plot of $\{\boldsymbol{z}_i\}_{i=1}^n$ can be easily visualized:



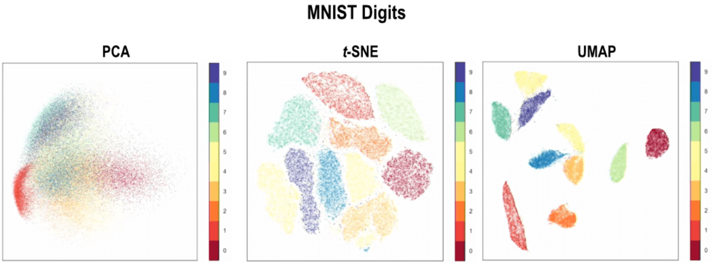
- The principal components $\{\boldsymbol{v}_j\}_{j=1}^R$ may also be meaningful ...

Example: PCA visualization of MNIST



Data visualization via t-SNE and UMAP

- To visualize high-dimensional data, it is best to use a nonlinear dimensionality reduction approach
- Classical methods include kernel PCA, IsoMap, and locally linear embedding, but they don't work very well
- The most popular methods today include t-stochastic neighbor embedding (t-SNE), uniform manifold approximation and projection (UMAP), and PaCMAP



Learning objectives

- Recognize need for feature dimensionality reduction
- Understand PCA as RSS-minimizing linear approximation
  - Understand orthogonal projection
  - Recognize PCA as subspace fitting
  - Understand the role of the data-covariance eigenvectors in PCA
  - Know how to measure PCA performance using PoV
- Understand how to compute PCA using the SVD
- Understand how the PCA coefficients can be used in supervised learning tasks
- Understand how PCA can be used for data visualization
- Be familar with t-SNE and UNET, non-linear data-visualization techniques