Lecture 11
Principal Component Analysis

EE-UY 4563 / EL-GY 9123: INTRODUCTION TO MACHINE LEARNING

PROF. SUNDEEP RANGAN

Outline

- Dimensionality reduction
- Principal components and directions of variance
- Approximation with PCs
- Computing PCs via the SVD
- Face example in python

Dimensionality Reduction

- Many modern data sets have very high dimension
- Want to reduce dimension:
  - Simplify classification / regression tasks on the data set
  - Visualize data
  - Find underlying commonalities in data

Data Definitions

- Given data: $\boldsymbol{x}_i, i = 1, \ldots, N$
- Each sample has p features: $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{ip})$
- 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?

Example: Faces

Labeled Faces in the Wild Home

- Face images can be high-dimensional
  - We will use 50 x 37 = 1850 pixels
- But, there may be few degrees of freedom
- Can we reduce the dimensionality of this?
- 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 10 years ago

Loading the Data

- ❑Lect10_PCA_Faces.ipynb
- ❑Built-in routines to load data is sciket-learn
- ❑Can take several minutes the first time (Be patient)

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

```python
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\Sundeep\scikit_learn_data\lfw_home\lfw_funneled
2016-11-14 14:22:08,605 Loading LFW people faces from C:\Users\Sundeep\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


```
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]])
```

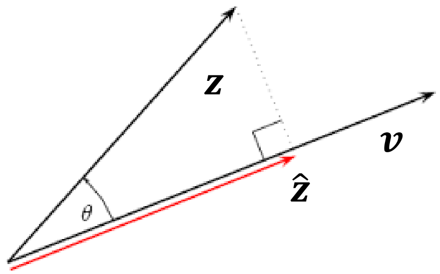
Outline

- Dimensionality reduction
- Principal components and directions of variance
- Approximation with PCs
- Computing PCs via the SVD
- Face example in python

Projections

- Given a vectors $\boldsymbol{z}$ and $\boldsymbol{v}$
- Projection of $\boldsymbol{z}$ onto $\boldsymbol{v}$ is:

$$\hat{\boldsymbol{z}} = \mathrm{Proj}_{\boldsymbol{v}}(\boldsymbol{z}) = \alpha \boldsymbol{v}, \qquad \alpha = \frac{\boldsymbol{v}^T\boldsymbol{z}}{\boldsymbol{v}^T\boldsymbol{v}} = \frac{\|\boldsymbol{z}\|}{\|\boldsymbol{v}\|} \cos\theta$$



- Let $V = \{\alpha\boldsymbol{v} | \alpha \in R\}$ = vectors on the line spanned by $\boldsymbol{v}$
- Theorem: $\mathrm{Proj}_{\boldsymbol{v}}(\boldsymbol{z})$ is closest point in V to $\boldsymbol{z}$:

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

Sample Covariance Matrix

- Let $\widetilde{\boldsymbol{X}}$ = data matrix with sample mean removed.
  - Rows: $\widetilde{\boldsymbol{x}}_i = \boldsymbol{x}_i - \overline{\boldsymbol{x}}$
- Sample covariance matrix: Matrix $\boldsymbol{Q}$ 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 $p \times p$
- Theorem: Sample covariance is given by

$$\boldsymbol{Q} = \frac{1}{N}\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{X}}$$

  - Proof on board
  - Compute sample covariance via a matrix product

Directional Variance

- Given data: $\boldsymbol{x}_i, i = 1, \ldots, N$ and direction $\boldsymbol{v}$ with $\|\boldsymbol{v}\| = 1$
- How much does $\boldsymbol{x}_i$ vary in the direction $\boldsymbol{v}$?
- Let $z_i = \boldsymbol{v}^T \boldsymbol{x}_i$ = projection of $\boldsymbol{x}_i$ onto $\boldsymbol{v}$
- Sample mean and variance in direction $\boldsymbol{v}$ is (proof on board):
  - Sample mean $\bar{z} = \boldsymbol{v}^T \overline{\boldsymbol{x}}$
  - Sample variance $s_z^2 = \boldsymbol{v}^T \boldsymbol{S} \boldsymbol{v}$

Maximizing Directional Variance

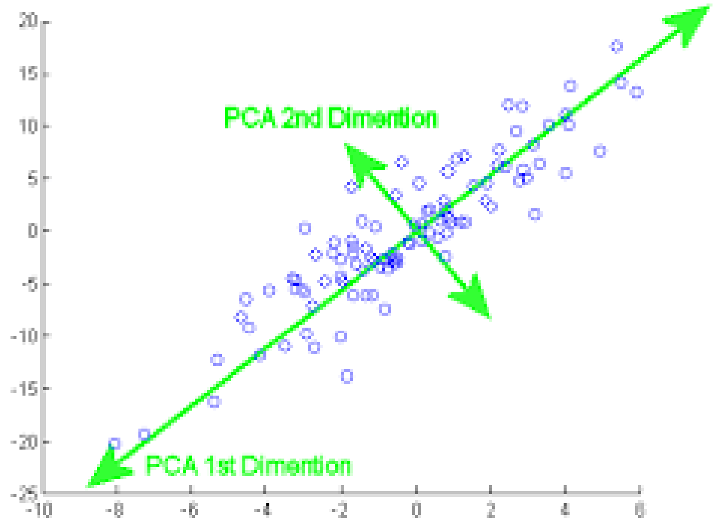
- What directions $\boldsymbol{v}$ maximize the variance?
- Formulate as an optimization problem:

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

- Let $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_p$ be the eigenvectors of $\boldsymbol{Q}$: $\boldsymbol{Q}\boldsymbol{v}_j = \lambda_j \boldsymbol{v}_j$
- Sort eigenvalues in descending order: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p$
  - Can show that eigenvalues are real and non-negative
- Theorem: Any local maxima of the variance directional is an eigenvector
  - $\boldsymbol{v} = \boldsymbol{v}_j$ for some j and $\boldsymbol{v}^{\boldsymbol{T}} \boldsymbol{Q} \boldsymbol{v} = \lambda_j$
  - Proof on board

Principal Components

- **Principal components**: The eigenvectors of $\boldsymbol{Q}$, $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_p$
  - Always normalized $\|\boldsymbol{v}_j\| = 1$
- Sorted by eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p$
- Each vector is of dimension p
- Key property: Vectors are orthogonal
  - $\boldsymbol{v}_j^T \boldsymbol{v}_k = 0$ if $j \neq k$
  - Proof on board
- Represents directions of maximal variance



Outline

- ❑Dimensionality reduction
- ❑Principal components and directions of variance
- ❑Approximation with PCs
- ❑Computing PCs via the SVD
- ❑Face example in python

Low-Dimensional Representations

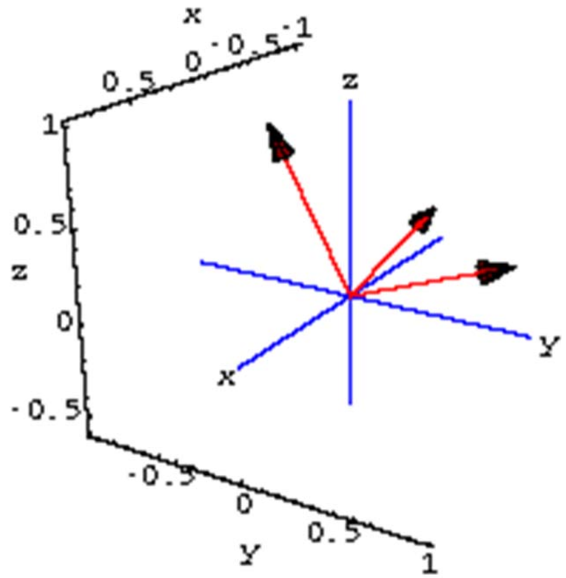
- Given data $\boldsymbol{x}_i, i = 1, \dots, N$
- Problem: Find basis vectors $\boldsymbol{v}_j, j = 1, \dots, d$ such that:

$$\boldsymbol{x}_i \approx \overline{\boldsymbol{x}} + \sum_{j=1}^{d} \alpha_{ij} \boldsymbol{v}_j$$

  - Sample mean + linear combination of basis vectors
  - $\alpha_i = (\alpha_{i1}, \dots, \alpha_{id})$ is an approximate coordinates of $\boldsymbol{x}_i$ in basis $(\boldsymbol{v}_1, \dots, \boldsymbol{v}_d)$
- Dimensionality reduction:
  - If $d \ll p$ we have represented $\boldsymbol{v}_i$ with a smaller number of coefficients.

Orthonormal Sets and Bases

- Definition: A set of vectors $\boldsymbol{v}_1, \dots, \boldsymbol{v}_d$ are an orthonormal set if:
  - $\|\boldsymbol{v}_j\| = 1$ for all j (unit length)
  - $\boldsymbol{v}_j^T \boldsymbol{v}_k = 0$ if $j \neq k$ (perpendicular to one another)
- If $d = p$ then $\boldsymbol{v}_1, \dots, \boldsymbol{v}_p$ is called an orthonormal basis
- Matrix form: If $\boldsymbol{V} = [\boldsymbol{v}_1 \ \dots \boldsymbol{v}_d]$, then $\boldsymbol{V}^T \boldsymbol{V} = I_d$
- If $d = p$, then $\boldsymbol{V}$ is an orthogonal matrix
- Key property: the PCs form an orthonormal basis



Coefficients in an Orthonormal Basis

- Suppose $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_p$ is an orthonormal basis
- Given a vector $\boldsymbol{z}$, can write

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

  - Simple expression for computing coefficients in an orthonormal basis
- Matrix form:

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

Approximating the Data Matrix

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

$$\boldsymbol{x}_i = \overline{\boldsymbol{x}} + \sum_{j=1}^{p} \alpha_{ij} \boldsymbol{v}_j, \qquad \alpha_{ij} = \boldsymbol{v}_j^T (\boldsymbol{x}_i - \overline{\boldsymbol{x}})$$

- Now consider approximation with d coefficients:

$$\widehat{\boldsymbol{x}}_i = \overline{\boldsymbol{x}} + \sum_{j=1}^{d} \alpha_{ij} \boldsymbol{v}_j$$

Average Approximation Error

- Let $\widehat{\boldsymbol{x}}_i$ = approximation with d PCs
- Error in sample i:

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

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

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

  - Sum of the smallest $p-d$ eigenvalues

Proportion of Variance (PoV)

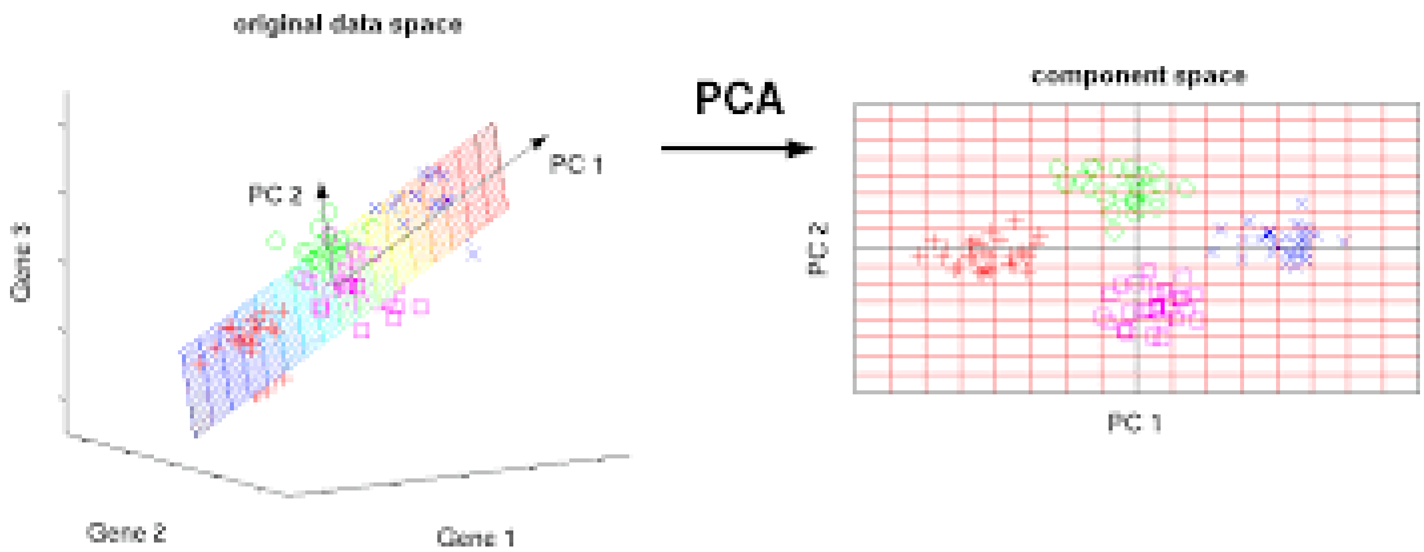
- Total variance of data set: $\frac{1}{\mathrm{N}}\sum_{i=1}^{N}\|\boldsymbol{x}_i - \overline{\boldsymbol{x}}\|^2 = \sum_{j=1}^{p}\lambda_j$
- Average approximation error: $\frac{1}{\mathrm{N}}\sum_{i=1}^{N}\|\boldsymbol{x}_i - \widehat{\boldsymbol{x}}_i\|^2 = \sum_{j=d+1}^{p}\lambda_j$
- The proportion 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: Suppose eigenvalues of sample covariance matrix are $10, 4, 0.2, 0.1, 0, 0, \ldots$
  - What is the PoV for $d = 1,2,3, \ldots$

Visualizing the Representation

- Finds a low-dimensional representation

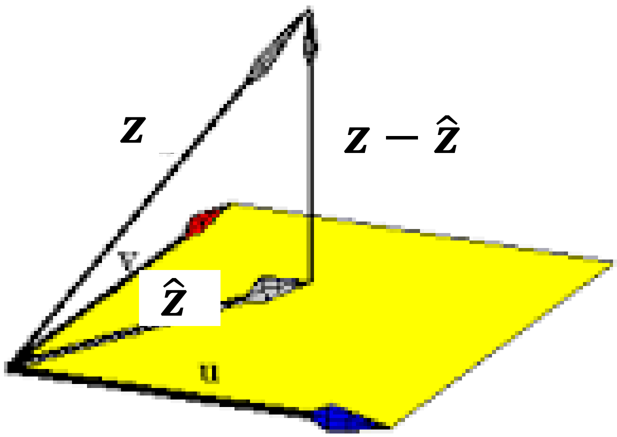


Geometry of Approximations

- Approximation can be interpreted geometrically
- Let V be set of all linear combinations

$$\sum_{j=1}^{d} \alpha_j \boldsymbol{v}_j$$

  - V is a vector space
  - Called the span of $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_d$
- Given $\boldsymbol{z}$, $\hat{\boldsymbol{z}}$ is the closest vector in V to $\boldsymbol{z}$
- Called the projection of $\boldsymbol{z}$ onto V

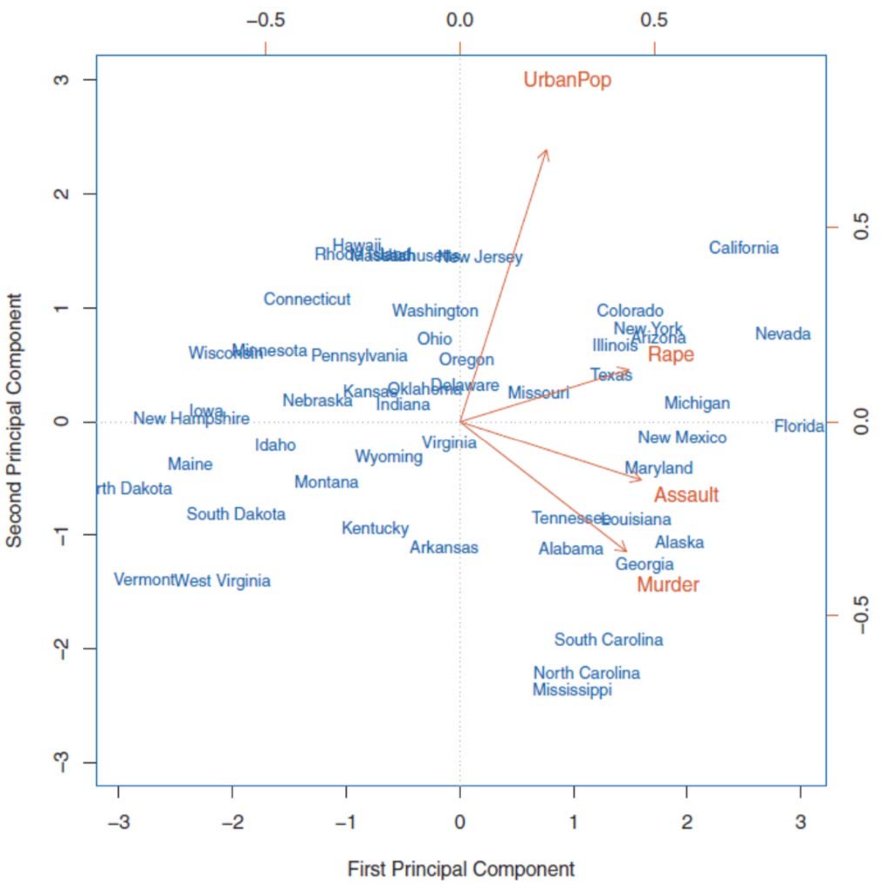


Space spanned by $v_1, \ldots, v_d$

Latent Representations

- Each record is of the form: $\boldsymbol{x}_i \approx \overline{\boldsymbol{x}} + \sum_{j=1}^{d} \alpha_{ij} \boldsymbol{v}_j$
- Variance in $\boldsymbol{x}_i$ explained by small number of "latent components"
  - Coefficients α_{ij} are the latent representations of $\boldsymbol{x}_i$
- Example:
  - $\boldsymbol{x}_i$ = 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

Outline

- ❑Dimensionality reduction
- ❑Principal components and directions of variance
- ❑Approximation with PCs
- ❑Computing PCs via the SVD
- ❑Face example in python

Singular Value Decomposition

- Given matrix $\boldsymbol{A} \in R^{M \times N}$
- SVD is $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T$, where
  - $\boldsymbol{U} \in R^{M \times r}$, $U^T U = I_r$
  - $\boldsymbol{V} \in R^{N \times r}$, $V^T V = I_r$
  - $\boldsymbol{S} = \mathrm{diag}(s_1, \dots, s_r)$, sorted $s_1 \geq s_2 \geq \cdots \geq s_r \geq 0$. Called the singular values
- All matrices have an SVD
- Number of singular values $r \leq \min(M, N)$

Computing the PCA via SVD

- Let $\tilde{\boldsymbol{X}}$ = data matrix with sample mean removed.
- Take SVD: $\tilde{\boldsymbol{X}} = \mathbf{USV}^T$
- Properties:
  - Sample covariance matrix is $\boldsymbol{Q} = \frac{1}{N}\tilde{\boldsymbol{X}}^T\tilde{\boldsymbol{X}} = \frac{1}{N}\boldsymbol{V}\boldsymbol{S}^2\boldsymbol{V}^T$
  - Eigenvalues are $\lambda_j = s_j^2/N$
  - PCs are $\boldsymbol{v}_j$, columns of $\boldsymbol{V}$
  - Coefficients are $\alpha = \tilde{\boldsymbol{X}}\boldsymbol{V} = \boldsymbol{US}$

Finding the Basis Vectors

- Consider problem of finding one basis vector, $\boldsymbol{v}$
- Given basis vector $\boldsymbol{u}$, the minimum approximation error for the i-th sample is:

$$\min_{\alpha_i} \|\boldsymbol{x}_i - \overline{\boldsymbol{x}} - \alpha_i \boldsymbol{v}\|^2$$

  - Represents how well $\boldsymbol{x}_i$ can be represented by the vector $\boldsymbol{v}$
- Define the average approximation error:

$$J(\boldsymbol{v}) := \frac{1}{N} \sum_{i=1}^{N} \min_{\alpha_i} \|\boldsymbol{x}_i - \overline{\boldsymbol{x}} - \alpha_i \boldsymbol{v}\|^2$$

- Select $\boldsymbol{v}$ to minimize approximation error

$$\min_{\boldsymbol{v}} J(\boldsymbol{v})$$

Outline

- ❑Dimensionality reduction
- ❑Principal components and directions of variance
- ❑Approximation with PCs
- ❑Computing PCs via the SVD
- ❑Face example in python

Computing the SVD

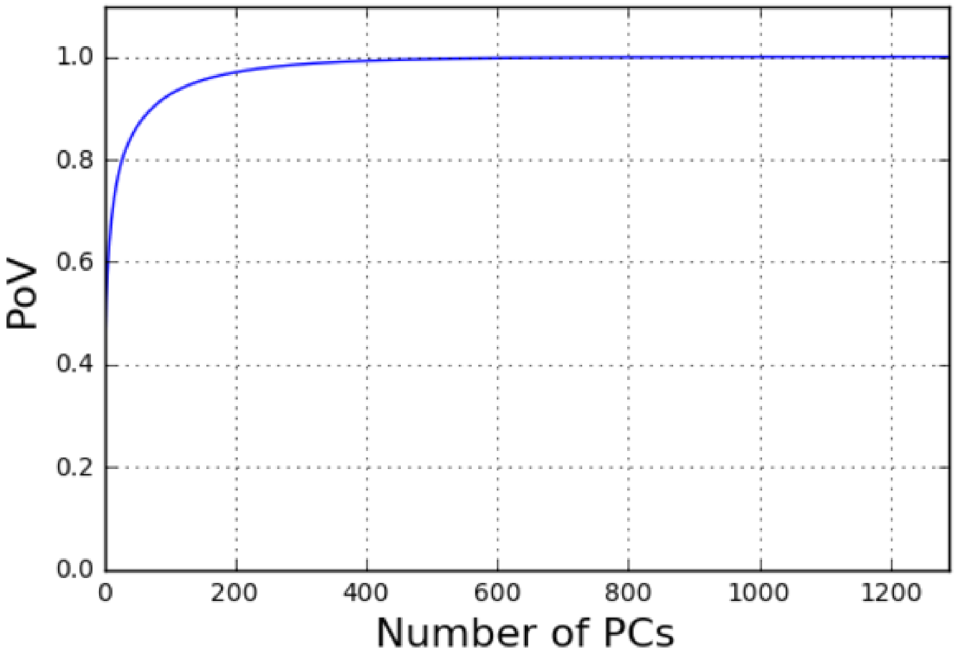
```
npix = h*w
Xmean = np.mean(X,0)
Xs = X - Xmean[None,:]
```

Then, we compute an SVD. Note that in python the SVD re

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

- Note efficient use of python broadcasting
- SVD:
  - Use full_matrices (avoids computing zero SVs)
  - Matrix V is what we call V^T
    (Different from MATLAB)

Finding the PoV



- Most variance explained in about 400 components
- Some reduction

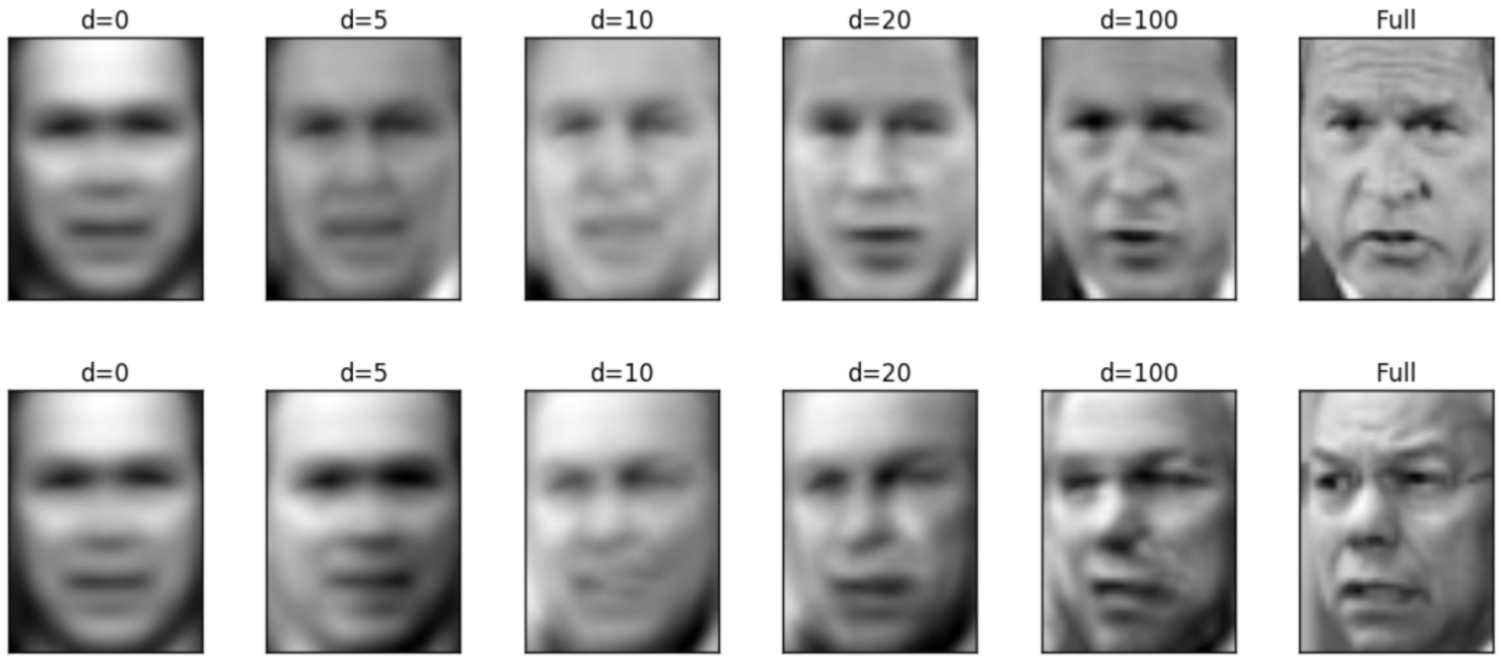
```
lam = S**2
PoV = np.cumsum(lam)/np.sum(lam)

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

# Select random faces
inds = np.random.permutation(n_samples)
inds = inds[:nplt]
nd = len(ds)

# Set figure size
plt.figure(figsize=(1.8 * (nd+1), 2.4 * nplt))
plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.35)

# Loop over figures
iplt = 0
for ind in inds:
    for d in ds:
        plt.subplot(nplt,nd+1,iplt+1)
        Xhati = (U[ind,:d]*S[None,:d]).dot(V[: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
```

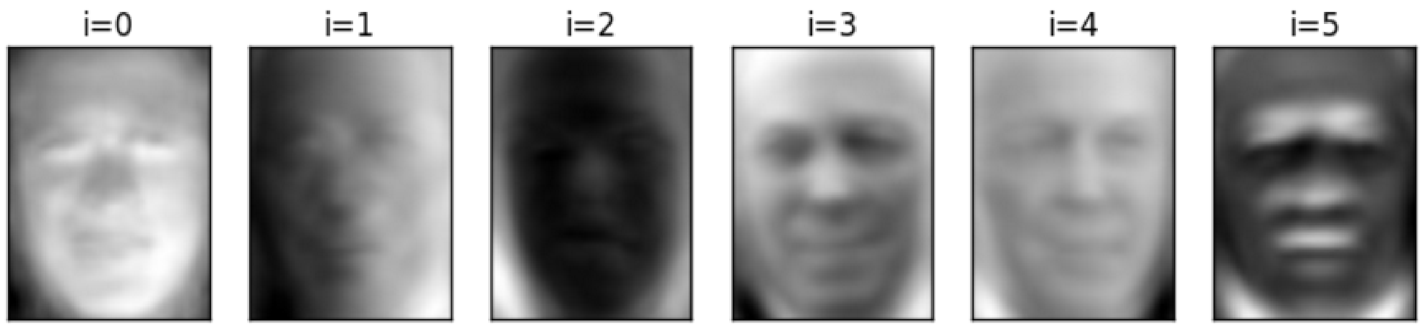
- Selection of figure sizes for subplot
- Note: Efficient computing of approx

Plotting the Approximations



Plotting the PCs

- The PCs can be plotted as well

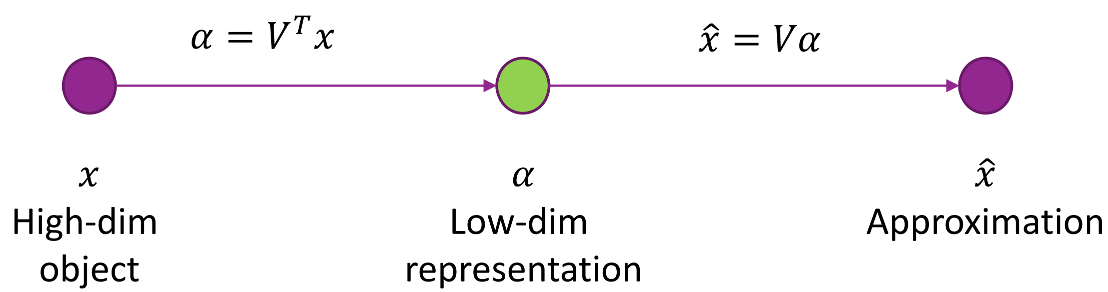


Other Considerations

- Normalization: For most data, it is essential to standardize before computing PC
  - Otherwise, components with large values dominate small ones
- Sklearn has built in PCA routine (will explore in lab)
- Some texts do not subtract mean
  - Will be picked up as one of the PCs

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



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)