FUNDAMENTALS OF MACHINE LEARNING

DIMENSIONALITY REDUCTION

CSCI3320

Prof. John C.S. Lui, CSE Department, CUHK
Introduction to Machine Learning

Why Reduce Dimensionality?

- Complexity of classifiers and regressors depend on the number & dimensions of inputs (N and d)
- Reduces time complexity: Less computation
- Reduces space complexity: Fewer parameters
- Saves the cost of observing the feature
- Simpler models are more robust on small datasets
- More interpretable: simpler explanation
- Data visualization (structure, groups, outliers, etc) if plotted in 2 or 3 dimensions

Feature Selection vs. Extraction

- **Feature selection:** Choosing $k < d$ important features, ignoring the remaining $d - k$

  Subset selection algorithms

- **Feature extraction:** Project the original $x_i, i = 1, .., d$ dimensions to **new** $k < d$ dimensions, $z_j, j = 1, .. k$
  - ***Principal Component Analysis (PCA), Factor Analysis (FA), Multidimensional Scaling (MDS)*** : linear projection and unsupervised method
  - ***Linear Discriminant Analysis (LDA)***: linear projection and supervised method
  - ***Isometric feature mapping (isomap), locality linear embedding (LLE)***: *nonlinear dimensionality reduction*

Subset Selection

- There are 2^d subsets of d features
- **Forward search:** Add the best feature at each step
  - Set of features *F* initially ∅.
  - At each iteration, find the best new feature

$$i = \text{argmin}_i E(F \cup x_i)$$

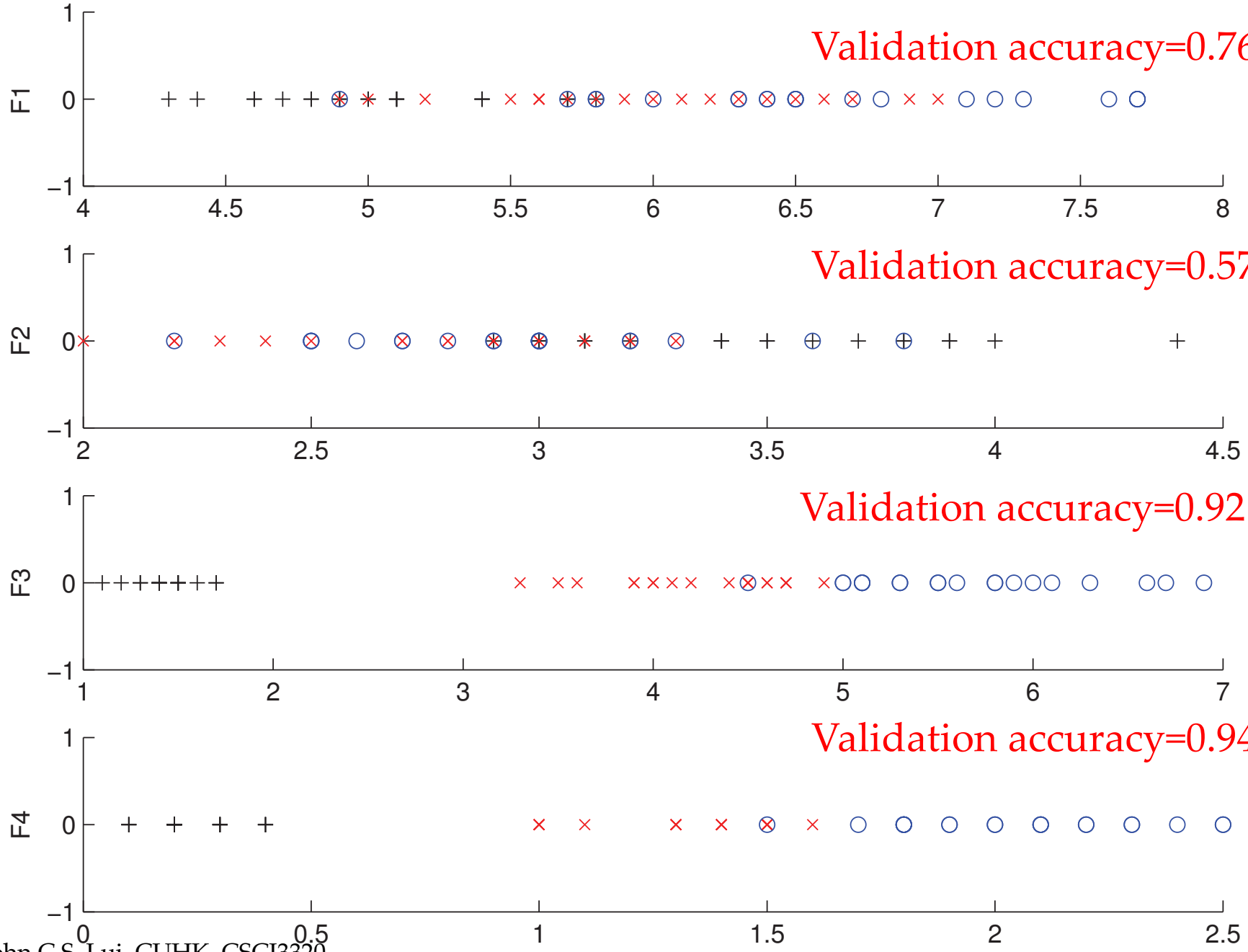
$$\text{Add } x_i \text{ to } F \text{ if } E(F \cup x_i) < E(F))$$

***E(F)*: error of using *F* during validation**

- Hill-climbing $O(d^2)$ algorithm, *greedy & **NOT** optimal !!*
- **Backward search:** Start with all features and remove one at a time, if possible.
- Floating search (Add k, remove l)

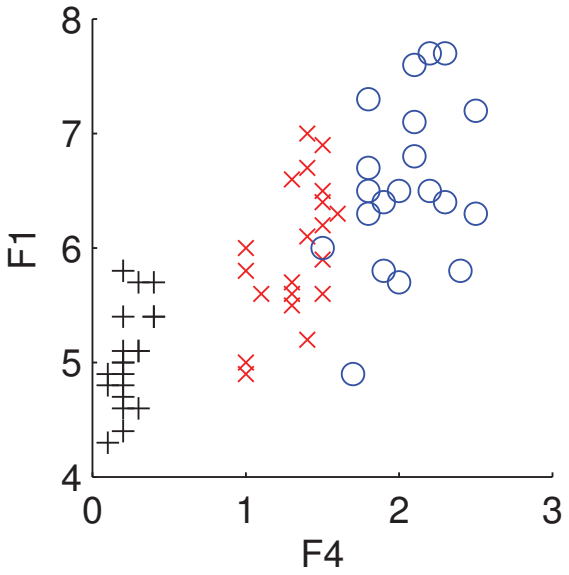
Forward is preferred than backward

Iris data: Single feature

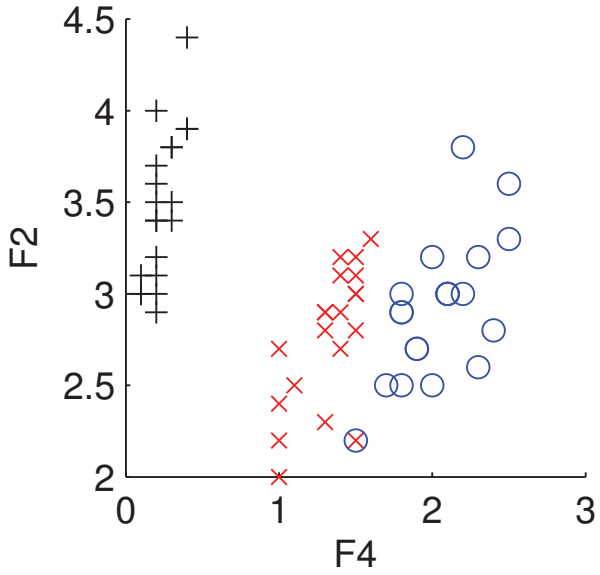


Iris data: Add one more feature to F4

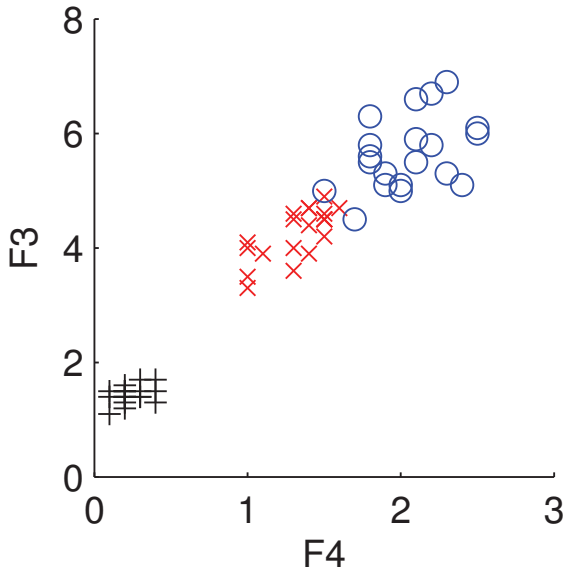
Validation accuracy=0.87



Validation accuracy=0.92



Validation accuracy=0.96



Chosen

Review: Eigenvectors and Eigenvalues

- Let ***A*** be an *nxn matrix*. We can λ and **x** are eigenvalue and eigenvector of ***A*** if

$$\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$$

Vector invariant in transformation

- Computing eigenvalue and eigenvector

$$(\boldsymbol{A} - \lambda\boldsymbol{I})\boldsymbol{x} = \boldsymbol{0} \Rightarrow (\boldsymbol{A} - \lambda\boldsymbol{I}) \text{ must } \mathbf{not} \text{ be invertible.}$$

- This condition means: $\det(\boldsymbol{A} - \lambda\boldsymbol{I}) = 0$

Review: Eigenvectors and Eigenvalues

Example

Let $A = \begin{bmatrix} 2 & -4 \\ -1 & -1 \end{bmatrix}$. Then $\begin{aligned} p(\lambda) &= \det \begin{bmatrix} 2-\lambda & -4 \\ -1 & -1-\lambda \end{bmatrix} \\ &= (2-\lambda)(-1-\lambda) - (-4)(-1) \\ &= \lambda^2 - \lambda - 6 \\ &= (\lambda - 3)(\lambda + 2). \end{aligned}$

Thus, $\lambda_1 = 3$ and $\lambda_2 = -2$ are the eigenvalues of A.

Let's find the eigenvectors corresponding to $\lambda_1 = 3$. Let $\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$. Then $(A - 3I)\mathbf{v} = \mathbf{0}$ gives us

$$\begin{bmatrix} 2-3 & -4 \\ -1 & -1-3 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

from which we obtain the duplicate equations

$$\begin{aligned} -v_1 - 4v_2 &= 0 \\ -v_1 - 4v_2 &= 0. \end{aligned}$$

Review: Eigenvectors and Eigenvalues

If we let $v_2 = t$, then $v_1 = -4t$. All eigenvectors corresponding to $\lambda_1 = 3$ are multiples of $\begin{bmatrix} -4 \\ 1 \end{bmatrix}$ and thus the eigenspace corresponding to $\lambda_1 = 3$ is given by the span of $\begin{bmatrix} -4 \\ 1 \end{bmatrix}$. That is, $\left\{ \begin{bmatrix} -4 \\ 1 \end{bmatrix} \right\}$ is a **basis** of the eigenspace corresponding to $\lambda_1 = 3$.

Repeating this process with $\lambda_2 = -2$, we find that

$$\begin{aligned} 4v_1 - 4v_2 &= 0 \\ -v_1 + v_2 &= 0 \end{aligned}$$

If we let $v_2 = t$ then $v_1 = t$ as well. Thus, an eigenvector corresponding to $\lambda_2 = -2$ is $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and the eigenspace corresponding to $\lambda_2 = -2$ is given by the span of $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$. $\left\{ \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}$ is a basis for the eigenspace corresponding to $\lambda_2 = -2$.

Note: we can also use Matlab or Python (and others) to find eigenvectors and eigenvalues

Example in Python (ev_example_3.py)

```
#! /usr/bin/env python

from numpy import linalg as LA

# define matrix A
A = ([[ 2.0,  -4.0],
      [ -1.0, -1.0]])

print ('Array is=', A)

# find eigvenvalues and eigenvectors
w,v = LA.eig(A)

print ('Eigenvalues are: ', w)
print ('Eigenvectors are:')
print (v)
```

Brief Introduction of Lagrange multiplier

- We want to maximize (or minimize) an objective function with ***equality*** constraints
- **Example 1:**

$$f(x,y) = 100x^{3/4}y^{1/4}$$

$$s.t. \quad 200x + 250y = 50,000$$

- Write this as Lagrange multiplier problem:

$$F(x,y,\lambda) = 100x^{3/4}y^{1/4} - \lambda(200x + 250y - 50,000)$$

- Find ***partial derivatives*** with *x*, *y* and λ and equate them to zero

Lagrange multiplier (continue)

- These partial derivative equations are:

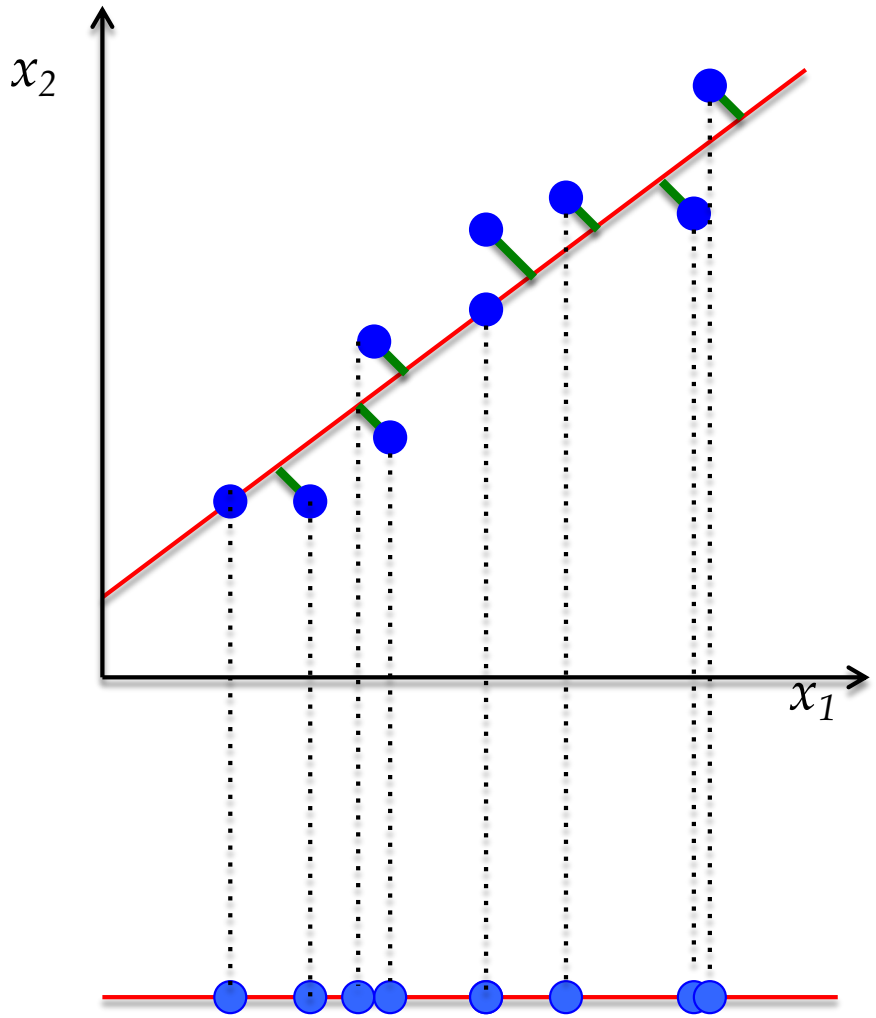
$$F_x(x, y, \lambda) = 75x^{-1/4}y^{1/4} - 200\lambda = 0$$

$$F_y(x, y, \lambda) = 25x^{3/4}y^{-3/4} - 250\lambda = 0$$

$$F_\lambda(x, y, \lambda) = -200x - 250y + 50,000 = 0$$

- Solve this system of three equations and three unknowns. To begin solve for λ in the first equation, substitute it in to the second equation to solve for x and substitute that into the final equation to solve for y.
- $x^*=187.5$, $y^*=50$, $\lambda^*=0.27$

Principal Components Analysis (PCA)



Reduction from 2D to 1D

$$\begin{array}{lcl} \boldsymbol{x}^{(1)} & \in & \mathbb{R}^2 \\ \boldsymbol{x}^{(2)} & \in & \mathbb{R}^2 \\ \boldsymbol{x}^{(3)} & \in & \mathbb{R}^2 \\ \vdots & \in & \vdots \\ \boldsymbol{x}^{(N)} & \in & \mathbb{R}^2 \end{array} \quad \Rightarrow \quad \begin{array}{lcl} z^{(1)} & \in & \mathbb{R} \\ z^{(2)} & \in & \mathbb{R} \\ z^{(3)} & \in & \mathbb{R} \\ \vdots & \in & \vdots \\ z^{(N)} & \in & \mathbb{R} \end{array}$$

Principal Components Analysis (PCA)

- Find a low-dimensional space such that when $\boldsymbol{x}$ is projected there, information loss is minimized.
- The projection of $\boldsymbol{x}$ on the direction of $\boldsymbol{w}$ is: $z = \boldsymbol{w}^T \boldsymbol{x}$
- Find $\boldsymbol{w}$ such that Var(z) **is maximized (why?)**

$$\begin{aligned}
\text{Var}(z) &= \text{Var}(\boldsymbol{w}^T\boldsymbol{x}) = E[(\boldsymbol{w}^T\boldsymbol{x} - \boldsymbol{w}^T\boldsymbol{\mu})^2] \\
&= E[(\boldsymbol{w}^T\boldsymbol{x} - \boldsymbol{w}^T\boldsymbol{\mu})(\boldsymbol{w}^T\boldsymbol{x} - \boldsymbol{w}^T\boldsymbol{\mu})] \\
&= E[\boldsymbol{w}^T(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T\boldsymbol{w}] \\
&= \boldsymbol{w}^T E[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T]\boldsymbol{w} = \boldsymbol{w}^T\Sigma\boldsymbol{w}
\end{aligned}$$

Because $\boldsymbol{w}^T(\boldsymbol{x} - \boldsymbol{\mu}) = (\boldsymbol{x} - \boldsymbol{\mu})^T\boldsymbol{w}$

where

$$\text{Var}(\boldsymbol{x}) = E[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T] = \Sigma \text{ (covariance matrix of } \boldsymbol{x})$$

- Maximize $\text{Var}(z)$ subject to $\|\boldsymbol{w}_1\| = 1$

$$\max_{\boldsymbol{w}_1} \boldsymbol{w}_1^T \Sigma \boldsymbol{w}_1 - \alpha(\boldsymbol{w}_1^T \boldsymbol{w}_1 - 1)$$

Lagrange multiplier

http://en.wikipedia.org/wiki/Lagrange_multiplier, example 1 and 2

- Take derivative with respect to $\boldsymbol{w}_1$, equate to 0, we have

$$2\Sigma \boldsymbol{w}_1 - 2\alpha \boldsymbol{w}_1 = 0 \qquad \Sigma \boldsymbol{w}_1 = \alpha \boldsymbol{w}_1$$

$\boldsymbol{w}_1$ is an eigenvector of Σ with eigenvalue α

Choose the one with the ***largest*** eigenvalue for $\text{Var}(z)$ to be maximized because

$$\boldsymbol{w}_1^T \Sigma \boldsymbol{w}_1 = \alpha \boldsymbol{w}_1^T \boldsymbol{w}_1 = \alpha$$

- Second principal component: Max Var(z_2), s.t., $\|\boldsymbol{w}_2\| = 1$ and orthogonal to $\boldsymbol{w}_1$ or $z_2 = \boldsymbol{w}_2^T\boldsymbol{x}$

$$\max_{\boldsymbol{w}_2} \boldsymbol{w}_2^T\boldsymbol{\Sigma}\boldsymbol{w}_2 - \alpha(\boldsymbol{w}_2^T\boldsymbol{w}_2 - 1) - \beta(\boldsymbol{w}_2^T\boldsymbol{w}_1 - 0)$$

- Taking derivative with $\boldsymbol{w}_2$:

$$2\boldsymbol{\Sigma}\boldsymbol{w}_2 - 2\alpha\boldsymbol{w}_2 - \beta\boldsymbol{w}_1 = 0$$

Similar property for other eigenvectors with decreasing eigenvalues

Premultiply by $\boldsymbol{w}_1^T$ and we get

$$2\boldsymbol{w}_1^T\boldsymbol{\Sigma}\boldsymbol{w}_2 - 2\alpha\boldsymbol{w}_1^T\boldsymbol{w}_2 - \beta\boldsymbol{w}_1^T\boldsymbol{w}_1 = 0$$

- Because $\boldsymbol{w}_1^T\boldsymbol{\Sigma}\boldsymbol{w}_2 = \boldsymbol{w}_2^T\boldsymbol{\Sigma}\boldsymbol{w}_1 = \lambda_1\boldsymbol{w}_2^T\boldsymbol{w}_1 = 0$

  we have: Then $\beta = 0$ and so $\boldsymbol{\Sigma}\boldsymbol{w}_2 = \alpha\boldsymbol{w}_2$

$\boldsymbol{w}_2$ eigenvector of $\boldsymbol{\Sigma}$ with **2nd largest eigenvalue** $\lambda_2 = \alpha$.

More on Σ

- Σ are symmetric, so all its eigenvectors are orthogonal
- When Σ are positive definite ($\boldsymbol{x}^T \Sigma \boldsymbol{x} > 0,$) then all its eigenvalues are positive
- If Σ is singular, then its rank (or effective dimensionality) is k with $k < d$ and

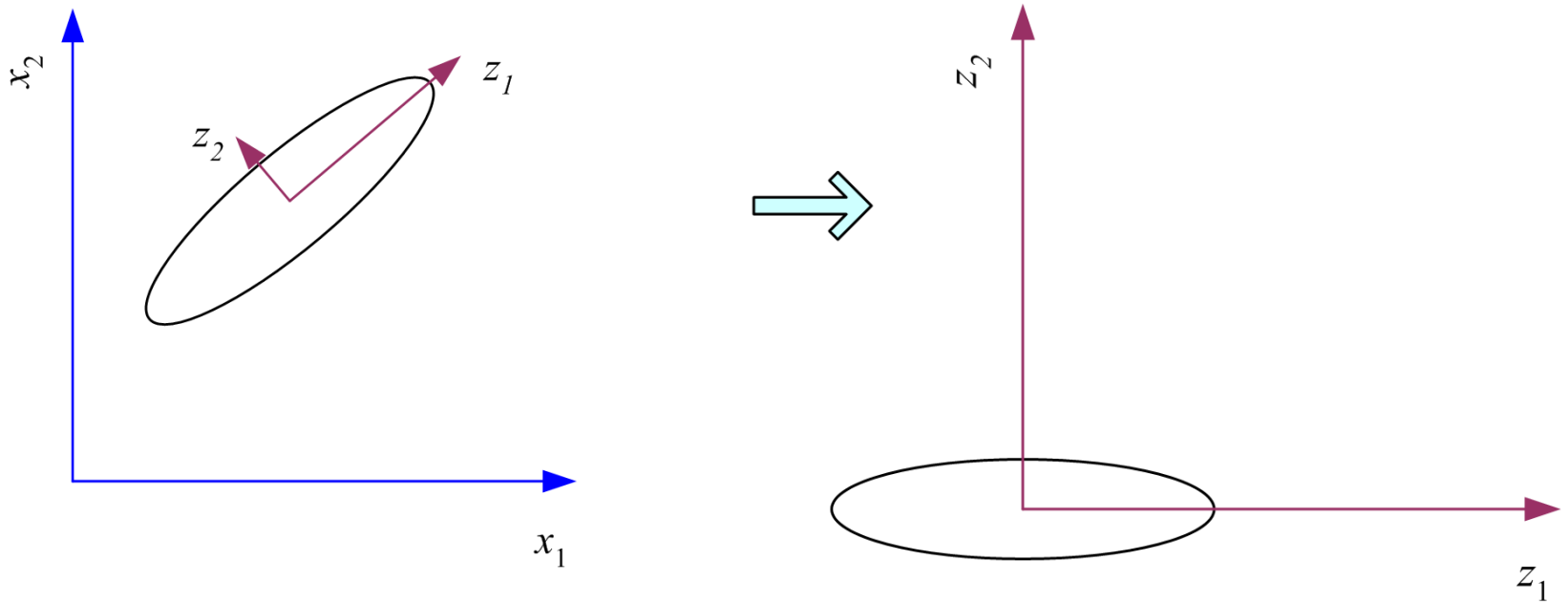
$$\lambda_i, i = k+1, \ldots, d \text{ are } 0$$

- These k eigenvectors and non-negative eigenvalues are the **dimensions of the reduced space**
- The first eigenvector, $\boldsymbol{w}_1$, the principle component, explains the largest part of the variance, and so on

What PCA does

$$\boldsymbol{z} = \boldsymbol{W}^T(\boldsymbol{x} - \boldsymbol{m}), \quad \boldsymbol{W} \text{ is a } dxk \text{ matrix}$$

where k columns of $\boldsymbol{W}$ are the eigenvectors of $\boldsymbol{S}$, **an estimate of** Σ, and subtract it from $\boldsymbol{m}$ (sample mean) before projection to center the data at the origin and rotates the axes

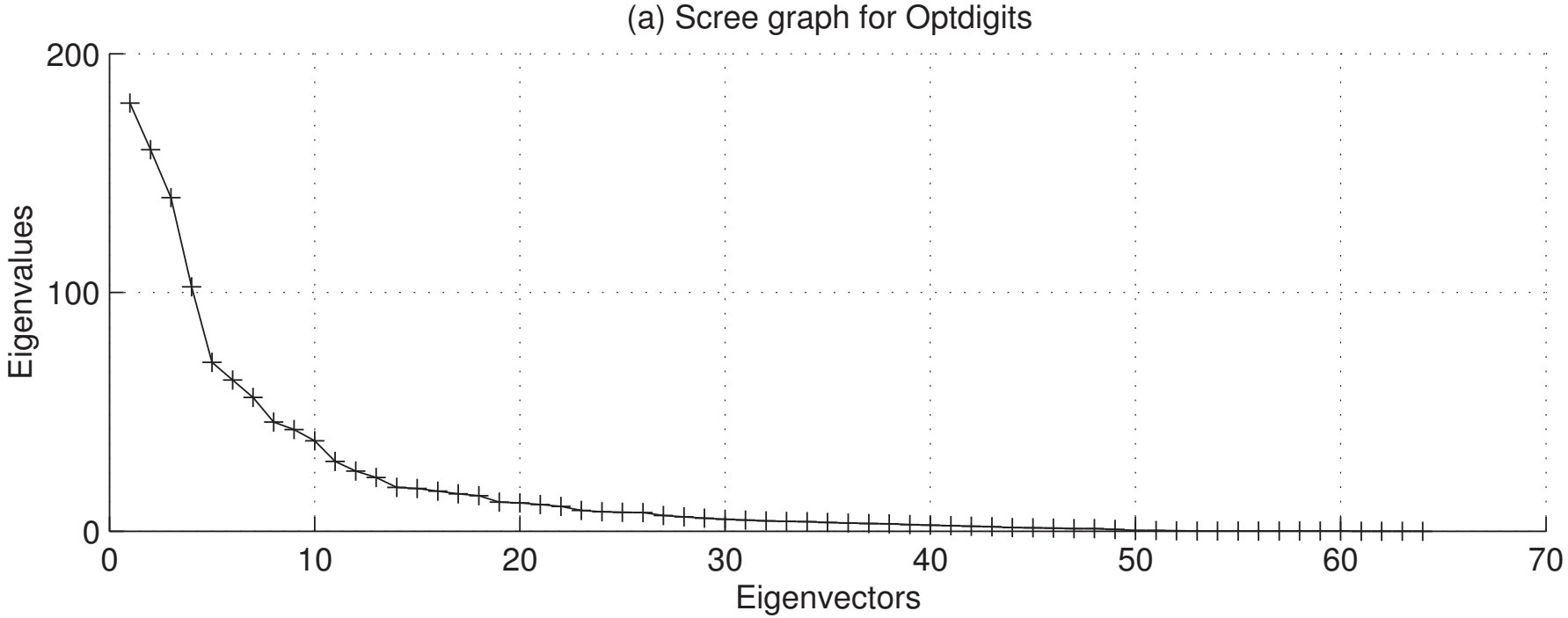


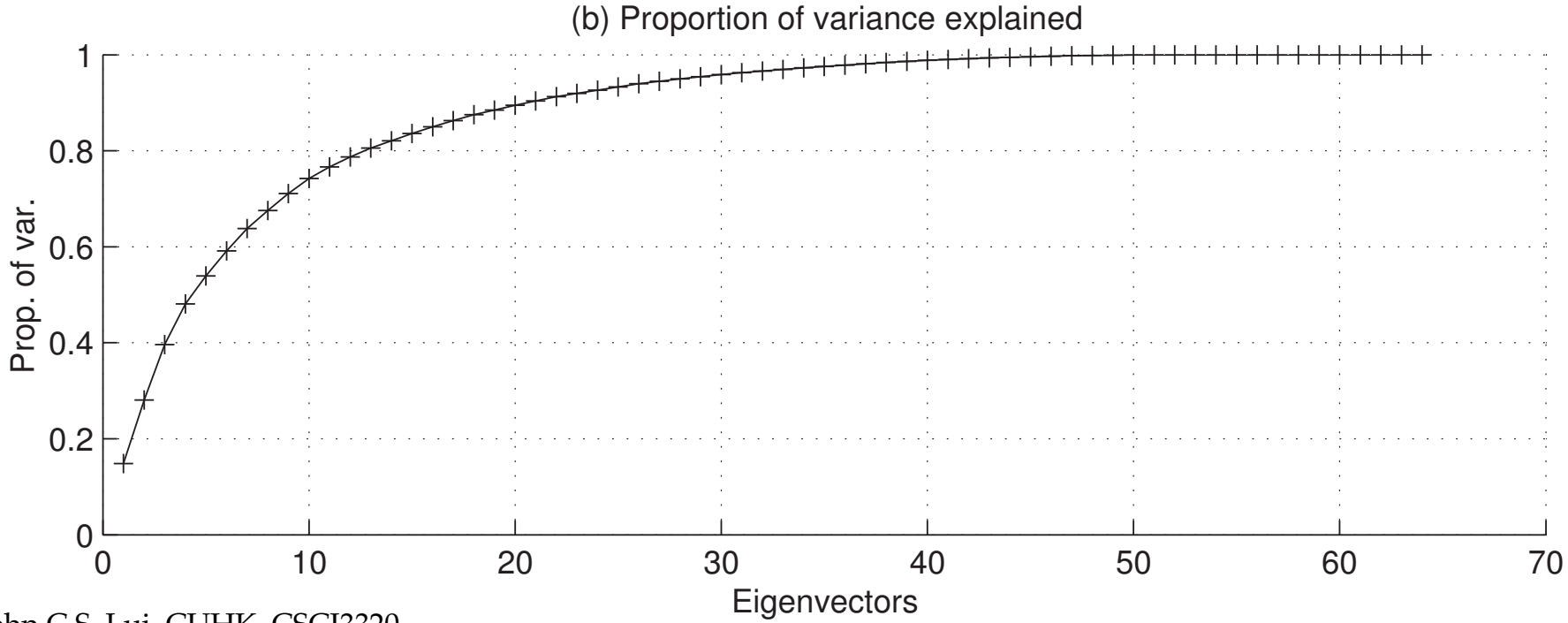
How to choose k ?

- **Proportion of Variance** (PoV) explained

$$\frac{\lambda_1 + \lambda_2 + \cdots + \lambda_k}{\lambda_1 + \lambda_2 + \cdots + \lambda_k + \cdots + \lambda_d}$$

  when λ_i are sorted in descending order
- Typically, stop at PoV>0.9
- If dimensions are highly correlated, we have few eigenvectors with large eigenvalues, and $k << d$
- Scree graph plots PoV vs k, stop at "elbow"


(a) Scree graph for Optdigits
Eigenvalues
200
100
0
0
10
20
30
40
50
60
70
Eigenvectors


(b) Proportion of variance explained
Prop. of var.
1
0.8
0.6
0.4
0.2
0
0
10
20
30
40
50
60
70
Eigenvectors

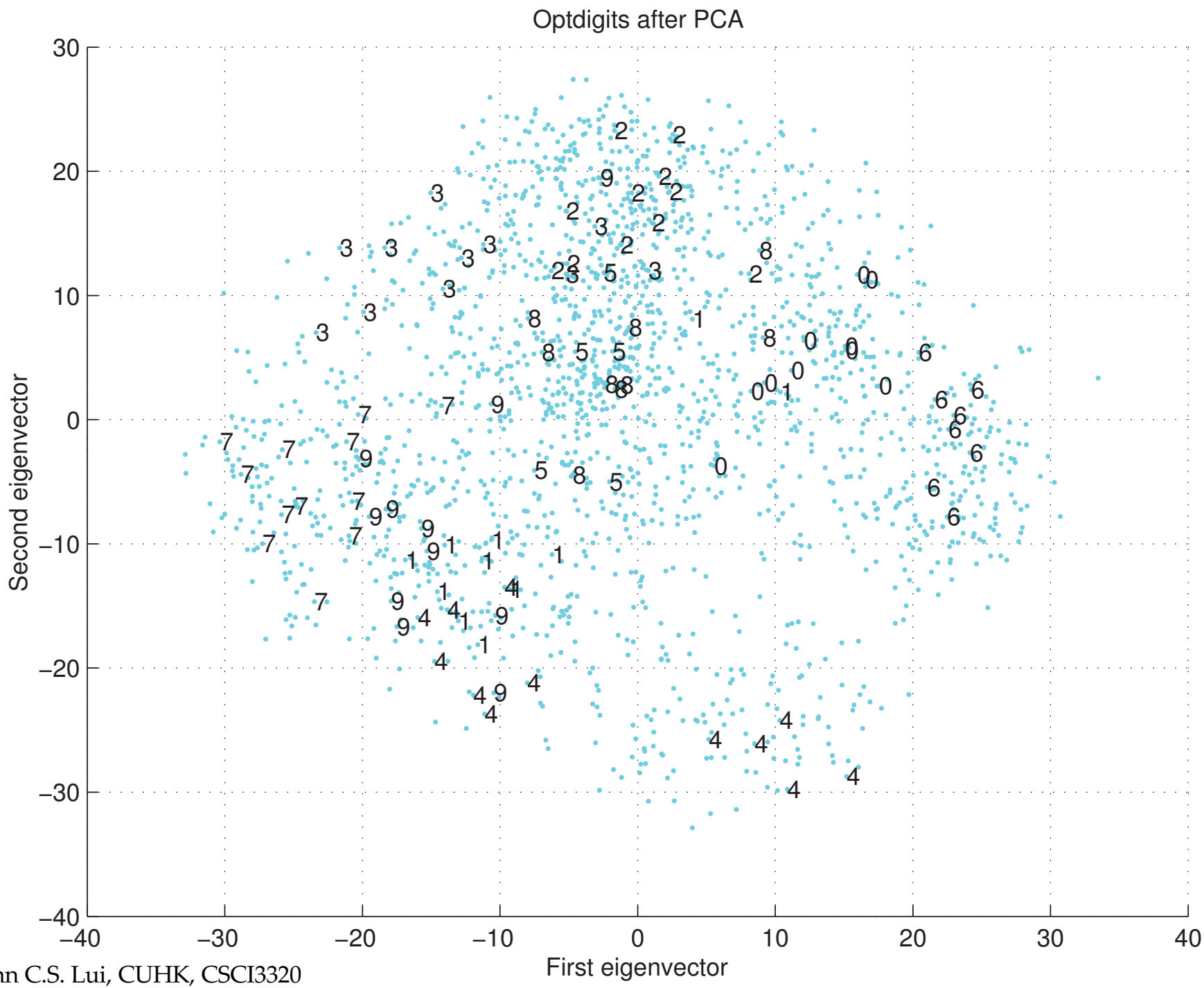

Optdigits after PCA
Second eigenvector
First eigenvector
30
20
10
0
−10
−20
−30
−40
−40
−30
−20
−10
0
10
20
30
40

Illustration via Python 3 (9 features, N=1000)

```
import numpy as np
from numpy import linalg as LA


x1  = np.random.normal(0.0, 1.0, (1000,1))        # x1 is N(0,1)
X1  = x1 - x1.sum()/1000.0 * np.ones_like(x1)     # X1 is mean centered
x2  = -1.0 * 4.0 *x1                              # x2 = -4*x1
X2  = x2 - x2.sum()/1000.0 * np.ones_like(x2)     # x2 is mean centered
x3  = 10.0 * x1 + 10.0                            # x3 = 10*x1 + 10
X3  = x3 - x3.sum()/1000.0 * np.ones_like(x3)     # x3 is mean centered

x4 = np.random.exponential(10.0, (1000,1))        # x4 is exp with mea = 10
X4 = x4 - x4.sum()/1000.0 * np.ones_like(x4)
x5 = -1.0 * 0.5 * x4                              # x5 is -x4/2.0
X5 = x5 - x5.sum()/1000.0 * np.ones_like(x5)
x6 = x4**2.0                                      # x6 is (x4)^2
X6 = x6 - x6.sum()/1000.0 * np.ones_like(x6)

x7 = np.random.uniform(-100.0, 100.0, (1000,1))   # x7 is uniform (-100,100)
X7 = x7 - x7.sum()/1000.0 * np.ones_like(x7)
x8 = -1.0*x7/10.0                                 # x8 is -0.1*x7
X8 = x8 - x8.sum()/1000.0 * np.ones_like(x8)
x9 = 2.0*x7 + 2.0                                 # x9 is 2*x7 + 2
X9 = x9 - x9.sum()/1000.0 * np.ones_like(x9)
```

Illustration via Python 3 (continue)

```
X = np.concatenate((X1,X2),axis=1)          # concatenate in column
X = np.concatenate((X,X3), axis=1)
X = np.concatenate((X,X4), axis=1)
X = np.concatenate((X,X5), axis=1)
X = np.concatenate((X,X6), axis=1)
X = np.concatenate((X,X7), axis=1)
X = np.concatenate((X,X8), axis=1)
X = np.concatenate((X,X9), axis=1)

print ('Shape of X is = ', X.shape)

# compute covariance matrix

X_input = X.T                    # X_input is the transpose of X
S = np.cov(X_input)              # get the covariance matrix
print('Shape of S = ', S.shape)

values, vectors = LA.eig(S)    # get eigenvalues/eigenvectors

print('eigenvalues: ', values)
print('eigenvectors: ', vectors)
```

Illustration via Python 3 (continue)

```
pc91061:Lecture Notes cslui1$ python eigen.py
Shape of X is =  (1000, 9)
Shape of S =  (9, 9)
eigenvalues:  [ 2.23026772e+05  1.65596193e+04  1.24426602e+02  2.64929173e+01
  8.12876073e-12  5.98841324e-13  1.66891353e-16 -2.29720026e-14
  3.04446443e-31]
eigenvectors:  [[ 1.52830135e-05 -7.94189709e-06 -9.24499562e-02  1.17718062e-04
   4.47872576e-02  9.88428406e-01 -4.86997504e-04  7.38704684e-04
   1.51337685e-18]
 [-6.11320539e-05  3.17675884e-05  3.69799825e-01 -4.70872249e-04
   9.29081151e-01 -7.80183096e-02 -1.58761319e-04  3.76806952e-04
  -6.29142303e-19]
 [ 1.52830135e-04 -7.94189709e-05 -9.24499562e-01  1.17718062e-03
   3.67153735e-01 -1.30050164e-01 -1.48047774e-05  7.68523125e-05
  -1.97350495e-19]
 [-1.92742581e-02 -9.47711615e-04 -1.14172778e-03 -8.94218263e-01
   2.54245460e-06  1.18856543e-03 -4.47169983e-01  6.25491463e-04
   1.35729230e-15]
 [ 9.63712903e-03  4.73855807e-04  5.70863890e-04  4.47109131e-01
   5.08490924e-06  2.37713087e-03 -8.94339967e-01  1.25098293e-03
   2.61270029e-15]
 [-9.99564419e-01 -2.01455048e-02 -1.36047687e-04  2.15664462e-02
   4.47271042e-16 -1.04325149e-16 -3.41249914e-18  5.27701701e-18
  -1.07119895e-18]
 [-9.00845538e-03  4.46676132e-01 -4.02163100e-05 -2.79174513e-04
  -1.34207230e-05  3.04340955e-05 -2.78839075e-04 -1.99800035e-02
  -8.94427191e-01]
 [ 9.00845538e-04 -4.46676132e-02  4.02163100e-06  2.79174513e-05
  -6.71036151e-04  1.52170478e-03 -1.39419537e-02 -9.99000174e-01
   9.51036087e-17]
 [-1.80169108e-02  8.93352263e-01 -8.04326200e-05 -5.58349026e-04
  -2.68414460e-05  6.08681911e-05 -5.57678150e-04 -3.99600070e-02
   4.47213595e-01]]
```

Illustration via Python 3: PCA

```
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(X)

print ('PCA values are: ', pca.singular_values_)

print ('PCA explain variance: ', pca.explained_variance_)
```

We see that we only need the 3 largest eigenvectors to capture most correlation !!!

Summary on the "operations" of PCA

- Given $\boldsymbol{X}$ (*N by d*) input matrix
- First, compute mean vector $\boldsymbol{m}$ (*d by 1*)
- Then normalize all inputs in $\boldsymbol{X}$ with $\boldsymbol{m}$ (subtraction of $\boldsymbol{m}$ in each row of $\boldsymbol{X}$)
- Compute Covariance matrix Σ from the mean normalized $\boldsymbol{X}$
- Perform PCA on Σ and decide the value of *k* via POV, this will result in a matrix $\boldsymbol{C}$ (*d by k*), with the first eigenvector in the 1st column, 2nd eigenvector in 2nd column, … and so on.
- Transform the mean normalized input $\boldsymbol{X}$ to a dimension-reduced input $\boldsymbol{Y}$ by $\boldsymbol{Y = XC}$ (*N by k matrix*)
- Use $\boldsymbol{Y}$ for logistic regression, parametric classification..
- For a new point $\boldsymbol{x}$, we mean normalized it, then $\boldsymbol{z=x^TC}$, fit $\boldsymbol{z}$ to our discriminant function or logistic regression

Final comment on PCA

- If variances of the original x_i *vary a lot, they affect the direction of the principle components than correlation*
- *To deal with this:*
  - *Preprocess the data so dimension has mean 0 and unit variance*
  - One may use eignevectors of the correlation matrix ***R*** instead of ***S***
- PCA explains variance and *it is sensitive to outliers*
- A few outliners can have large effect on the variances and thus the eigenvectors
- To deal with few outliners: *Compute Mahalanobis distance of the input points, discard the isolated data points that are far away*

Final comment on PCA

Read textbook

- Eigenfaces and eigendigits for images analysis
- If *d* is large, processing ***S*** is tedious. It is possible to calculate eigenvectors and eigenvalues directly from data without calculating the covariance matrix
- Reconstruction error
- PCA is unsupervised and does not use output or class information.
- **Karlunen-Loeve (KL)** expansion uses class information
- **Further explanation of PCA**:
  http://math.stackexchange.com/questions/23596/why-is-the-eigenvector-of-a-covariance-matrix-equal-to-a-principal-component

Deriving PCA via Spectral Decomposition

- Find W (d by k) such that $\underset{kx1}{z} = W^T \underset{dx1}{x}$ and Cov (z) = D, where D is any diagonal matrix (or uncorrelated z_i)
- We form a d x d matrix C where the i^{th} column is the normalized eigenvector c_i of S, then $C^TC = CC^T = I$ and

$$\begin{aligned} S &= SCC^T \\ &= S(c_1, c_2, \ldots, c_d)C^T \\ &= (Sc_1, Sc_2, \ldots, Sc_d)C^T \\ &= (\lambda_1 c_1, \lambda_2 c_2, \ldots, \lambda_d c_d)C^T \\ &= \lambda_1 c_1 c_1^T + \cdots + \lambda_d c_d c_d^T \\ &= CDC^T \end{aligned}$$

$$\begin{aligned} C^TC &= I \\ CC^TC &= CI = C \\ CC^T &= C(C)^{-1} = I \end{aligned}$$

$$S = CDC^T$$

- D is a diagonal matrix with eigenvalues $\lambda_1, \ldots, \lambda_d$
- This is the ***spectral decomposition*** of S

PCA via Spectral Decomposition

- Since **C** is orthogonal, so $\boldsymbol{C^TC = CC^T = I}$
- We then left multiply $\boldsymbol{C^T}$ and right multiply by **C**

$$\mathbf{C}^T\mathbf{S}\mathbf{C} = \mathbf{D}$$

- We know that if $\boldsymbol{z = W^Tx}$, then Cov (**z**) = $\boldsymbol{W^TSW}$, which we like to be a diagonal matrix.

Refer to page 14

- Compare the two expressions, it is easy to see that

$$\boldsymbol{W = C}$$

- Cite **example** in book

Feature Embedding

- When $\boldsymbol{X}$ is the Nxd data matrix (N data pts, d attributes)

$\boldsymbol{X}^T\boldsymbol{X}$ is the dxd matrix (covariance of features, *if mean-centered*)

$\boldsymbol{X}\boldsymbol{X}^T$ is the NxN matrix (*pairwise similarities of instances*)

- PCA uses the **eigenvectors** of $\boldsymbol{X}^T\boldsymbol{X}$ ***(or centered S)***, which are d-dimensional and can be used for projection
- Remember, spectral decomposition is: $\boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{W}\boldsymbol{D}\boldsymbol{W}^T$, $\boldsymbol{W}$ is dxd and contains eigenvector of $\boldsymbol{X}^T\boldsymbol{X}$ in its columns and $\boldsymbol{D}$ is dxd diagonal matrix with eigenvalues along the diagonal (*assume sorted*)
- If $\boldsymbol{X}^T\boldsymbol{X}$ has a rank $k < d$, then $D_{ii}=0$ for $i > k$.
- In PCA, we reduce to d dimension by taking the first k columns of $\boldsymbol{W}$. We denote them by $\boldsymbol{w}_i$ with eigenvalue λ_i.

Example of X^TX and XX^T

Let $N = 3$, $d = 2$, and $\boldsymbol{X} = \begin{bmatrix} x_1^1 & x_2^1 \\ x_1^2 & x_2^2 \\ x_1^3 & x_2^3 \end{bmatrix}$, $\boldsymbol{X}^T = \begin{bmatrix} x_1^1 & x_1^2 & x_1^3 \\ x_2^1 & x_2^2 & x_2^3 \end{bmatrix}$

$$\boldsymbol{X}^T\boldsymbol{X} = \begin{bmatrix} x_1^1 * x_1^1 + x_1^2 * x_1^2 + x_1^3 * x_1^3 & x_1^1 * x_2^1 + x_1^2 * x_2^2 + x_1^3 * x_2^3 \\ x_2^1 * x_1^1 + x_2^2 * x_1^2 + x_2^3 * x_1^3 & x_2^1 * x_2^1 + x_2^2 * x_2^2 + x_2^3 * x_2^3 \end{bmatrix}$$

$$\boldsymbol{X}\boldsymbol{X}^T = \begin{bmatrix} x_1^1 * x_1^1 + x_2^1 * x_2^1 & x_1^1 * x_1^2 + x_2^1 * x_2^2 & x_1^1 * x_1^3 + x_2^1 * x_2^3 \\ x_1^2 * x_1^1 + x_2^2 * x_2^1 & x_1^2 * x_1^2 + x_2^2 * x_2^2 & x_1^2 * x_1^3 + x_2^2 * x_2^3 \\ x_1^3 * x_1^1 + x_2^3 * x_2^1 & x_1^3 * x_1^2 + x_2^3 * x_2^2 & x_1^3 * x_1^3 + x_2^3 * x_2^3 \end{bmatrix}$$

Feature Embedding

- We map **x** to the new k-dimensional space via

$$z_i^t = \boldsymbol{w}_i^T \boldsymbol{x}^t, \;\; i = 1, \ldots, k, t = 1, \ldots, N$$

- Given that λ_i and $\boldsymbol{w}_i$ are the eigenvalues and eigenvectors of $\boldsymbol{X}^T\boldsymbol{X}$ for any *i* <= *k*, we have

$$\begin{aligned} (\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{w}_i &= \lambda_i \boldsymbol{w}_i \\ (\boldsymbol{X}\boldsymbol{X}^T)\boldsymbol{X}\boldsymbol{w}_i &= \lambda_i \boldsymbol{X}\boldsymbol{w}_i \end{aligned}$$

- So $\boldsymbol{X}\boldsymbol{w}_i$ are eigenvectors of $\boldsymbol{X}\boldsymbol{X}^T$ with the same eigenvalues
- Remember $\boldsymbol{X}^T\boldsymbol{X}$ is a *dxd* matrix, $\boldsymbol{X}\boldsymbol{X}^T$ is an *NxN* matrix, using spectral decomposition of $\boldsymbol{X}\boldsymbol{X}^T$

$$\boldsymbol{X}\boldsymbol{X}^T = \boldsymbol{V}\boldsymbol{E}\boldsymbol{V}^T$$

- **V** is the *NxN* matrix of eigenvectors of $\boldsymbol{X}\boldsymbol{X}^T$, **E** is an *NxN* diagonal matrix with corresponding eigenvalues

Feature Embedding

- $\boldsymbol{V}$, the NxN matrix, is the N-dimensional eigenvectors of $\boldsymbol{XX}^T$, called the *feature embedding*
- Feature embedding uses the eigenvectors of $\boldsymbol{XX}^T$, which are N-dim and which give directly the coordinates after projection
- When $d < N$, it is simpler to work with $\boldsymbol{X}^T\boldsymbol{X}$ (**or PCA**).
- When $d > N$, it is simpler to work with $\boldsymbol{XX}^T$ (**or feature embedding**)

Factor Analysis *(reverse of PCA)*

- Find a small number of **latent factors z**, which when combined generate **x** :

$$x_i - \mu_i = v_{i1}z_1 + v_{i2}z_2 + \cdots + v_{ik}z_k + \epsilon_i, \forall i = 1, ...d$$

$$x_i - \mu_i = \sum_{j=1}^{k} v_{ij}z_j + \epsilon_i$$

Example of latent factors: laziness, dumb, irresponsible,…

In matrix form: $\boldsymbol{x} - \boldsymbol{\mu} = \boldsymbol{V}\boldsymbol{z} + \boldsymbol{\epsilon}$

z_j, $j = 1, ..., k$ are *unit normal* **latent factors** with

$E[z_j] = 0$, $\mathrm{Var}(z_j) = 1$, $\mathrm{Cov}(z_i, z_j) = 0, i \neq j$

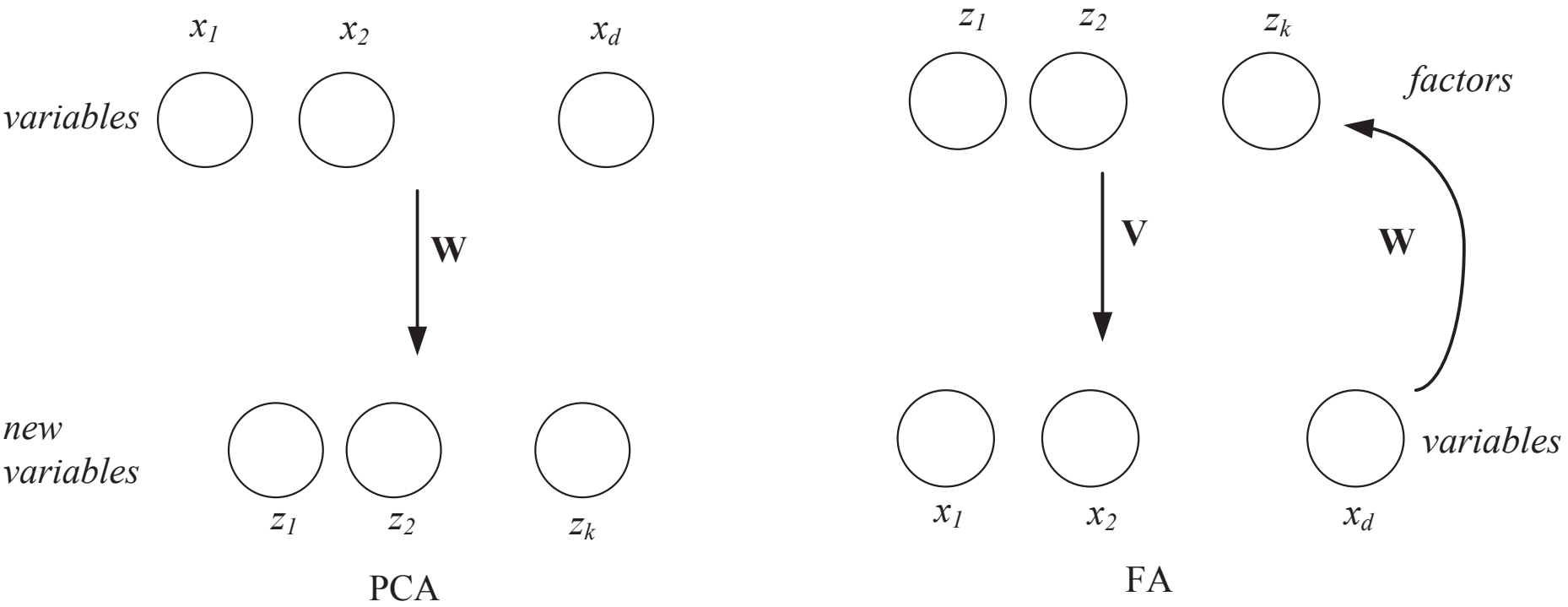
ϵ_i are **noise sources**:

$E[\epsilon_i] = 0$, $\mathrm{Var}(\epsilon_i) = \psi_i$, $\mathrm{Cov}(\epsilon_i, \epsilon_j) = 0, i \neq j$

and v_{ij} are the **factor loadings**

PCA vs FA

- PCA is from $\mathbf{x}$ to $\mathbf{z}$: $\mathbf{z} = \mathbf{W}^T(\mathbf{x} - \boldsymbol{\mu})$
- FA is from $\mathbf{z}$ to $\mathbf{x}$: $\mathbf{x} - \boldsymbol{\mu} = \mathbf{V}\mathbf{z} + \boldsymbol{\varepsilon}$



Factor Analysis

- Since we have: $x_i - \mu_i = v_{i1}z_1 + v_{i2}z_2 + \cdots + v_{ik}z_k + \epsilon_i, \forall i = 1, \ldots, d$
- Assume $\mu = 0$. Since $\text{Var}(z_j) = 1$ and $\text{Var}(\epsilon_i) = \psi_i$

$$\text{Var}(x_i) = v_{i1}^2 + v_{i2}^2 + \cdots + v_{ik}^2 + \psi_i \quad (*)$$

- In matrix form, we have

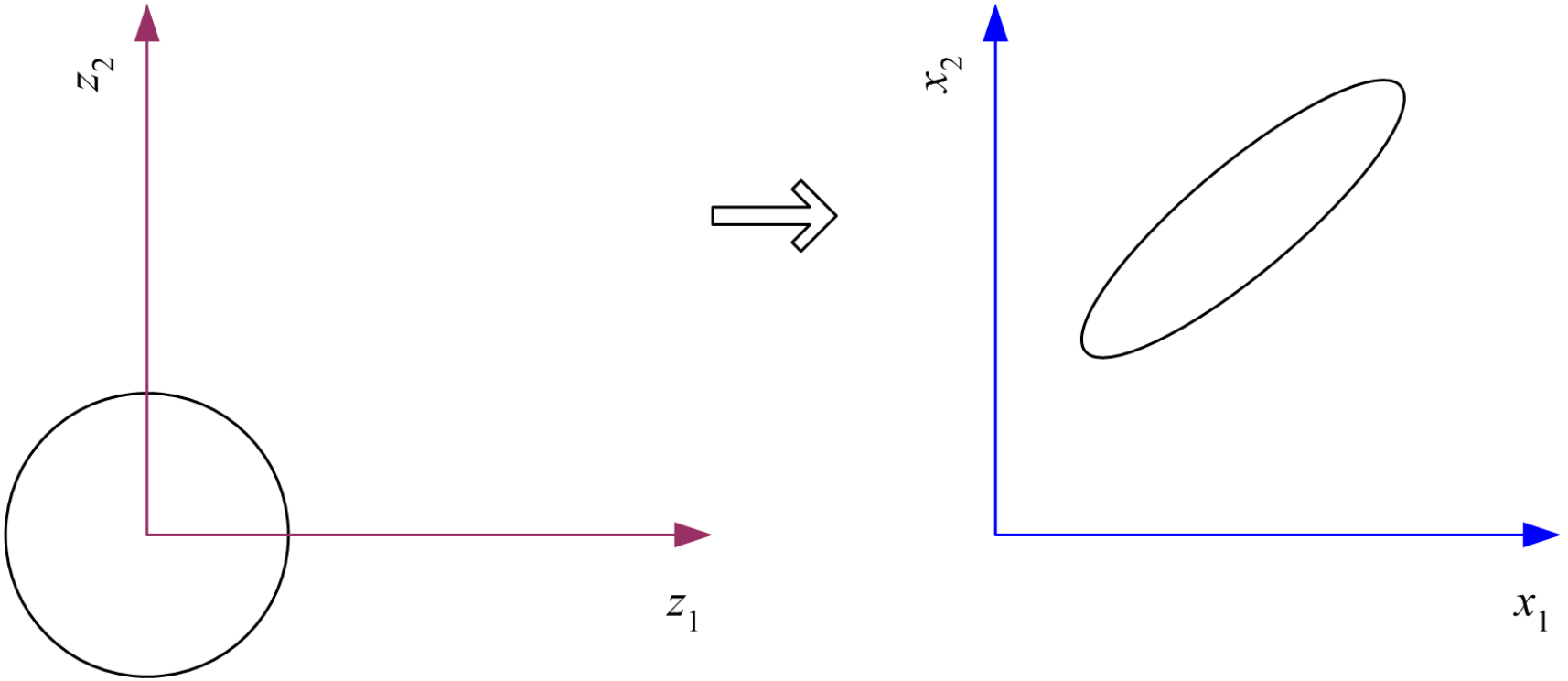
$$\begin{aligned} \boldsymbol{\Sigma} = \text{Cov}(\boldsymbol{x}) &= \text{Cov}(\mathbf{V}\boldsymbol{z} + \boldsymbol{\epsilon}) \\ &= \text{Cov}(\mathbf{V}\boldsymbol{z}) + \text{Cov}(\boldsymbol{\epsilon}) \\ &= \mathbf{V}\text{Cov}(\boldsymbol{z})\mathbf{V}^T + \boldsymbol{\Psi} \\ &= \mathbf{V}\mathbf{V}^T + \boldsymbol{\Psi} \end{aligned}$$

$\boldsymbol{\Psi}$ is a diagonal matrix with ψ_i on the diagonals.

$\text{Cov}(\boldsymbol{z}) = \mathbf{I}$ because factors are uncorrelated unit normal

Factor Analysis

- In FA, factors z_i are stretched, rotated and translated to generate **x**



Factor Analysis

- The question is: how to find ***V*** and ***Z***
- We have: $\Sigma = VV^T + \Psi$
- Ignore Ψ for now and focus on ***V***
- Using *spectral decomposition*, we have

$$S = CDC^T = CD^{1/2}D^{1/2}C^T = (CD^{1/2})(CD^{1/2})^T$$

C is $d \times k$ matrix of eigenvectors of S
$D^{1/2}$ is $k \times k$ diagonal matrix with square root of eigenvalues of S

- By inspection, we have: $\boxed{V = CD^{1/2}}$
- For Ψ , refer page 37 *(*)*, we have: $\psi_i = s_i^2 - \sum_{j=1}^{k} v_{ij}^2$
- For **Z**, a *Nxk matrix (refer book)*: $\boxed{Z = XS^{-1}V}$

Singular Value Decomposition and Matrix Factorization

- Given the *N x d* data matrix ***X***
- Singular value decomposition: $X=VAW^T$
  - ***V*** is *NxN* orthogonal matrix and contains the eigenvectors of XX^T *(an NxN matrix)*
  - ***W*** is *dxd* orthogonal matrix and contains the eigenvectors of X^TX *(an dxd matrix)*
  - **A** is *Nxd* and contains singular values on its first $k=min(N,d)$ diagonal that are square root of the nonzero eigenvalues of both XX^T and X^TX, we have

$$XX^T = (VAW^T)(VAW^T)^T = VAW^TWA^TV^T = VAA^TV = VEV^T$$
$$X^TX = (VAW^T)^T(VAW^T) = WA^TV^TVAW^T = WA^TAW^T = WDW^T$$

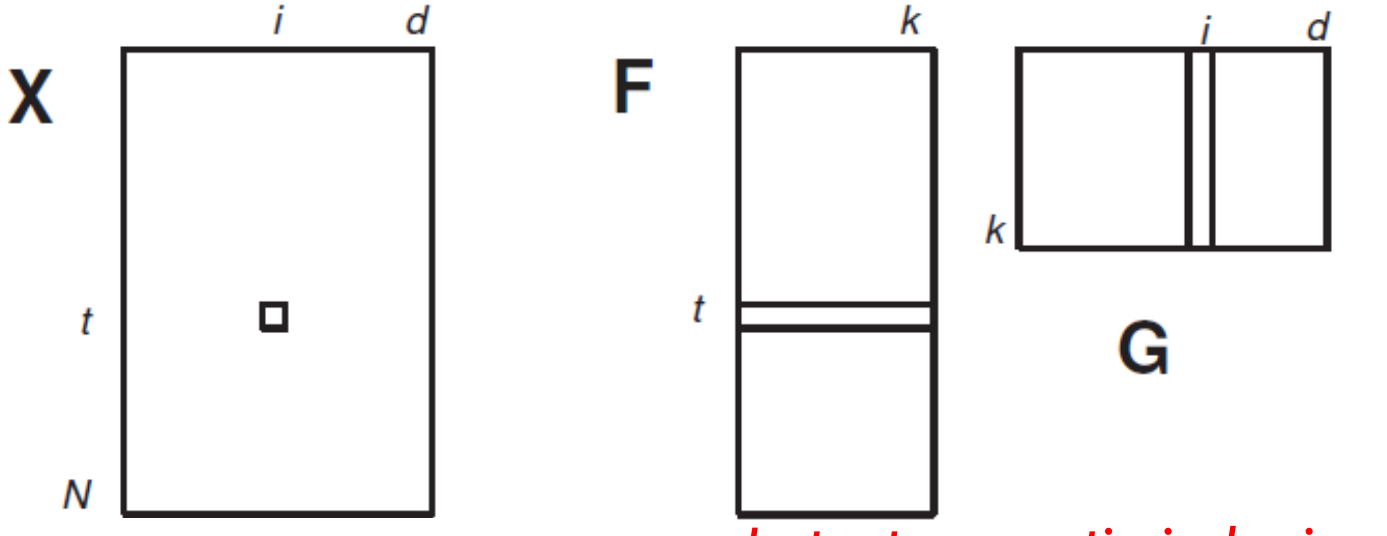
- $X=u_1a_1v_1^T+...+u_ka_kv_k^T$ where *k* is the rank of ***X***.

  Can ignore u_i and v_i for very small a_i

Matrix Factorization

- Matrix factorization: rewrite ***X*** as product of two matrices: ***X=FG***

  X is *N*x*d*, ***F*** is *N*x*k* and ***G*** is *k*x*d*, if *k*<< *N*,*d*, storing **F** and **G** reduce space!!!!



$$\mathbf{X}_{ti} = \mathbf{F}_t^T \mathbf{G}_i = \sum_{j=1}^{k} \mathbf{F}_{tj} \mathbf{G}_{ji}$$

Latent semantic indexing

Matrix Factorization: *X=FG*

- ***X*** is *Nxd*, ***F*** is *Nxk* and ***G*** is *kxd*, if *k<< N,d*, storing **F** and ***G*** reduce space!!!!
- ***F*** defines data instances in terms of *k* hidden factors
- ***G*** defines factors "in terms" of the original *k* attributes
- Example of ***X***: "bags of *d* words" for *N* documents, each factor may be one topic using a subset of words. So each document is some combination of such factor (or topic).
- Another example: N customers' purchase

$$\mathbf{X}_{ti} = \mathbf{F}_t^T \mathbf{G}_i = \sum_{j=1}^{k} \mathbf{F}_{tj} \mathbf{G}_{ji}$$

Latent semantic indexing

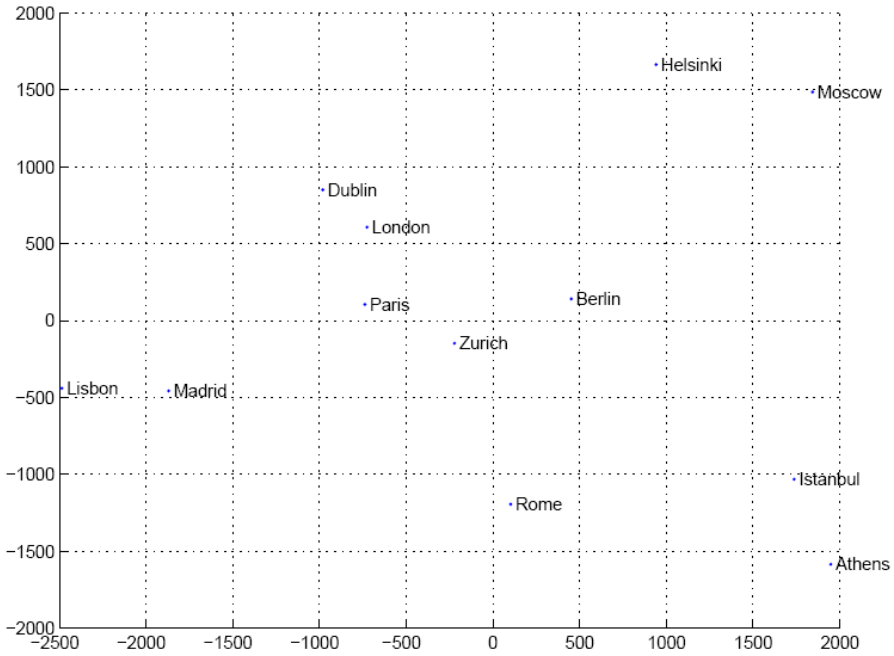
Multidimensional Scaling (MDS)

- Given pairwise distances between N points, d_{ij}, $i,j=1,..N$
- Place all these N points on a low-dimension map s.t. distances are preserved
- **MDS** can be used as dimension reduction by calculating pairwise Euclidean distances in the d-dimensional $\boldsymbol{x}$ space
- **PCA** done on correlation matrix equals to MDS with standardized Euclidean distances, each variable has unit variance
- $\boldsymbol{z} = \boldsymbol{g}(\boldsymbol{x} \mid \vartheta)$ Find ϑ that min Sammon stress

$$E(\theta|\mathcal{X}) = \sum_{r,s} \frac{(\|\boldsymbol{z}^r - \boldsymbol{z}^s\| - \|\boldsymbol{x}^r - \boldsymbol{x}^s\|)^2}{\|\boldsymbol{x}^r - \boldsymbol{x}^s\|^2}$$

$$= \sum_{r,s} \frac{(\|\boldsymbol{g}(\boldsymbol{x}^r|\theta) - \boldsymbol{g}(\boldsymbol{x}^s|\theta)\| - \|\boldsymbol{x}^r - \boldsymbol{x}^s\|)^2}{\|\boldsymbol{x}^r - \boldsymbol{x}^s\|^2}$$

Map of Europe by MDS



Map from CIA – The World Factbook: http://www.cia.gov/

Linear Discriminant Analysis (LDA)

- LDA a **supervised method** for dimensionality reduction
- Find a low-dimensional space such that when ***x*** is projected, ***classes are well-separated***
- We start with K=2, then K>2 classes
- Given two classes C_1 and C_2, find the direction ***w*** such that the data projected onto ***w***, the two classes are well separated.
- Projection of ***x*** onto ***w*** is: $z = \boldsymbol{w}^T \boldsymbol{x}$ (from *d* dimension to *one dimension*)
- Mean of class C_i sample **before** project: $\boldsymbol{m}_i \in \Re^d$
- Mean of class C_i sample **after** project: $m_i \in \Re$

Linear Discriminant Analysis (LDA)

- Given $\mathcal{X} = \{\boldsymbol{x}^t, r^t\}$ such that $r^t = 1$ if $\boldsymbol{x}^t \in C_1$ and $r^t = 0$ if $\boldsymbol{x}^t \in C_2$
- **Means after project:**

$$m_1 = \frac{\sum_t \boldsymbol{w}^T \boldsymbol{x}^t r^t}{\sum_t r^t} = \boldsymbol{w}^T \boldsymbol{m}_1$$

$$m_2 = \frac{\sum_t \boldsymbol{w}^T \boldsymbol{x}^t (1 - r^t)}{\sum_t (1 - r^t)} = \boldsymbol{w}^T \boldsymbol{m}_2$$

- **The scatter of samples from these two classes:**

$$s_1^2 = \sum_t (\boldsymbol{w}^T \boldsymbol{x}^t - m_1)^2 r^t$$

$$s_2^2 = \sum_t (\boldsymbol{w}^T \boldsymbol{x}^t - m_2)^2 (1 - r^t)$$

- To well-separated these two classes after projection, we want:

$$\text{large} |m_1 - m_2| \quad \text{small} \ s_1^2 + s_2^2$$

Linear Discriminant Analysis (LDA)

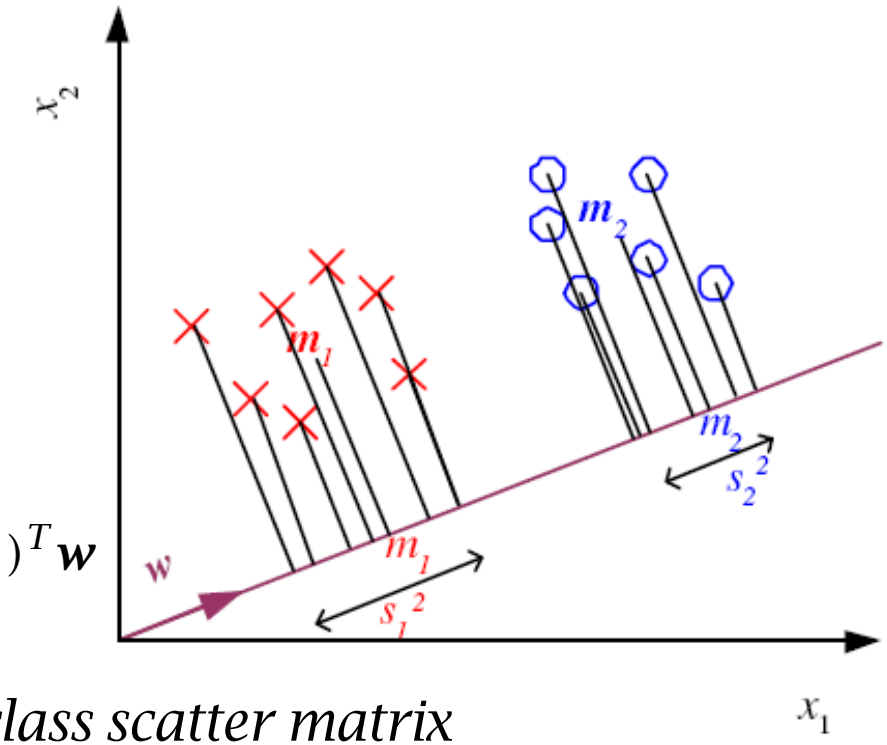
- Find $\boldsymbol{w}$ that maximizes

  (or Fisher's Linear Discriminant)

$$J(\boldsymbol{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2}$$

- Derivation:

$$\begin{aligned}(m_1 - m_2)^2 &= (\boldsymbol{w}^T\boldsymbol{m}_1 - \boldsymbol{w}^T\boldsymbol{m}_2)^2 \\ &= \boldsymbol{w}^T(\boldsymbol{m}_1 - \boldsymbol{m}_2)(\boldsymbol{m}_1 - \boldsymbol{m}_2)^T\boldsymbol{w} \\ &= \boldsymbol{w}^T\mathbf{S}_B\boldsymbol{w}\end{aligned}$$



$\mathbf{S}_B = (\boldsymbol{m}_1 - \boldsymbol{m}_2)(\boldsymbol{m}_1 - \boldsymbol{m}_2)^T$ *between-class scatter matrix*

$$\begin{aligned}s_1^2 &= \sum_t (\boldsymbol{w}^T\boldsymbol{x}^t - m_1)^2 r^t \\ &= \sum_t \boldsymbol{w}^T(\boldsymbol{x}^t - \boldsymbol{m}_1)(\boldsymbol{x}^t - \boldsymbol{m}_1)^T\boldsymbol{w}r^t \\ &= \boldsymbol{w}^T\mathbf{S}_1\boldsymbol{w}\end{aligned}$$

where

$$\mathbf{S}_1 = \sum_t r^t(\boldsymbol{x}^t - \boldsymbol{m}_1)(\boldsymbol{x}^t - \boldsymbol{m}_1)^T$$

within-class scatter matrix

Linear Discriminant Analysis (LDA)

- Similarly, we have

$$s_2^2 = \boldsymbol{w}^T \mathbf{S}_2 \boldsymbol{w} \text{ with } \mathbf{S}_2 = \sum_t (1 - r^t)(\boldsymbol{x}^t - \boldsymbol{m}_2)(\boldsymbol{x}^t - \boldsymbol{m}_2)^T$$

- So: $s_1^2 + s_2^2 = \boldsymbol{w}^T \mathbf{S}_W \boldsymbol{w}$ where $\mathbf{S}_W = \mathbf{S}_1 + \mathbf{S}_2$
- Finally, we want to find $\boldsymbol{w}$ which maximizes

$$J(\boldsymbol{w}) = \frac{\boldsymbol{w}^T \mathbf{S}_B \boldsymbol{w}}{\boldsymbol{w}^T \mathbf{S}_W \boldsymbol{w}} = \frac{|\boldsymbol{w}^T (\boldsymbol{m}_1 - \boldsymbol{m}_2)|^2}{\boldsymbol{w}^T \mathbf{S}_W \boldsymbol{w}}$$

- Take derivative of J with respect to $\boldsymbol{w}$ and equate to 0:

$$\frac{\boldsymbol{w}^T (\boldsymbol{m}_1 - \boldsymbol{m}_2)}{\boldsymbol{w}^T \mathbf{S}_W \boldsymbol{w}} 2 \left((\boldsymbol{m}_1 - \boldsymbol{m}_2) - \frac{\boldsymbol{w}^T (\boldsymbol{m}_1 - \boldsymbol{m}_2)}{\boldsymbol{w}^T \mathbf{S}_W \boldsymbol{w}} \mathbf{S}_W \boldsymbol{w} \right) = 0$$

- Given that $\boldsymbol{w}^T (\boldsymbol{m}_1 - \boldsymbol{m}_2) / \boldsymbol{w}^T \mathbf{S}_W \boldsymbol{w}$ is a constant, we have

LDA solution: $\boldsymbol{w} = c \mathbf{S}_W^{-1} (\boldsymbol{m}_1 - \boldsymbol{m}_2)$

Only magnitude is important, so we set c=1

Linear Discriminant Analysis (LDA)

- LDA (or **Fisher's Linear Discriminant**):

$$\boldsymbol{w} = \mathbf{S}_W^{-1}(\boldsymbol{m}_1 - \boldsymbol{m}_2)$$

- When $p(\boldsymbol{x}|C_i) \sim \mathcal{N}(\boldsymbol{\mu}_i, \Sigma)$, we have

$$\boldsymbol{w} = \Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$$

$K > 2$ Classes

- We want to the matrix $\boldsymbol{W}$ (dxk) such that $\boldsymbol{z} = \boldsymbol{W}^T \boldsymbol{x}$
- The within-class scatter matrix for C_i

$$\mathbf{S}_i = \sum_t r_i^t (\boldsymbol{x}^t - \boldsymbol{m}_i)(\boldsymbol{x}^t - \boldsymbol{m}_i)^T \quad \text{where } r_i^t = 1 \text{ if } \boldsymbol{x}^t \in C_i \text{ and 0 otherwise}$$

- The total within-class scatter is $\mathbf{S}_W = \sum_{i=1}^{K} \mathbf{S}_i$
- For $K > 2$, the scatter of means & between-class matrix:

$$\boldsymbol{m} = \frac{1}{K} \sum_{i=1}^{K} \boldsymbol{m}_i \qquad \mathbf{S}_B = \sum_{i=1}^{K} N_i (\boldsymbol{m}_i - \boldsymbol{m})(\boldsymbol{m}_i - \boldsymbol{m})^T \quad \text{with } N_i = \sum_t r_i^t$$

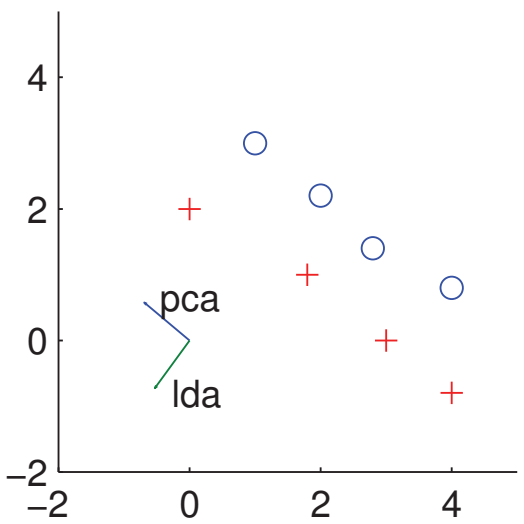
- The between-class & within-class scatter matrices (both are of dimension *k x k)after projection:*

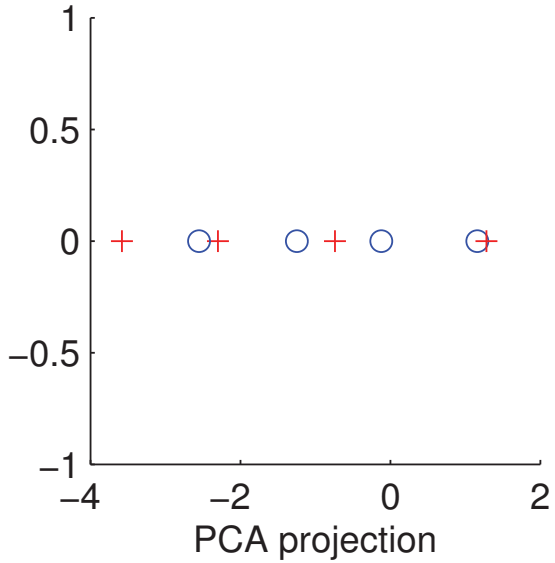
$\mathbf{W}^T \mathbf{S}_B \mathbf{W}$ **want large** $\mathbf{W}^T \mathbf{S}_W \mathbf{W}$ **want small**

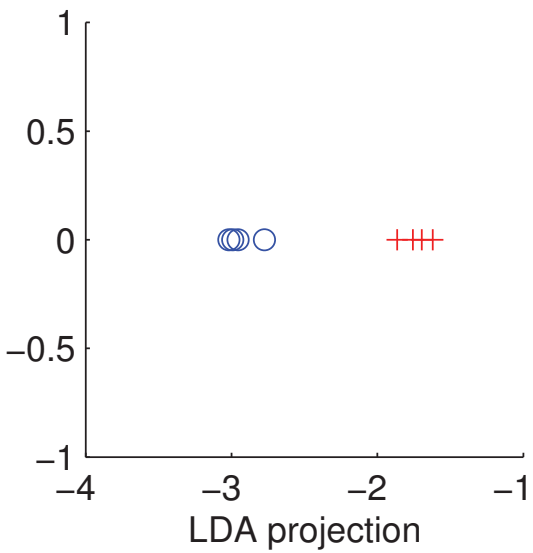
$K > 2$ Classes

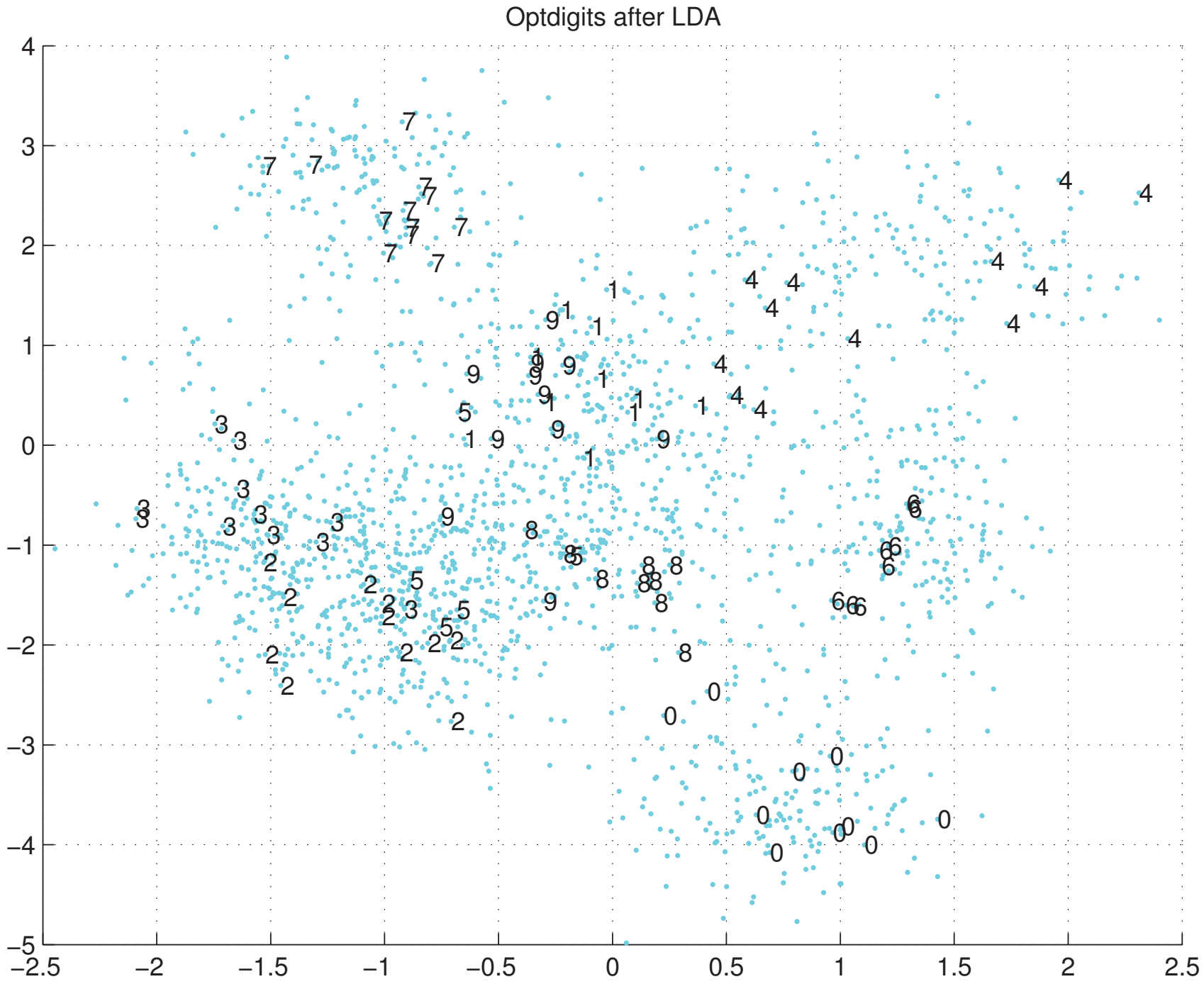
- Find matrix **W** that maximizes $J(\mathbf{W}) = \frac{|\mathbf{W}^T \mathbf{S}_B \mathbf{W}|}{|\mathbf{W}^T \mathbf{S}_W \mathbf{W}|}$
- The largest $K-1$ eigenvector of $\boldsymbol{S}_W^{-1}\boldsymbol{S}_B$ are the solution because $\boldsymbol{S}_B$ has a maximum rank of $K-1$
- Note that to use LDA, $\boldsymbol{S}_w$ has to be invertible. If not, we can first use PCA to get rid of singularity, then use LDA. Also we need to make sure PCA does not over reduce the dimensionality such that LDA has nothing to work on.
- LDA uses class information, it usually performs better than PCA

PCA vs LDA (2-dimension data)








Optdigits after LDA
4
3
2
1
0
−1
−2
−3
−4
−5
−2.5
−2
−1.5
−1
−0.5
0
0.5
1
1.5
2
2.5

Canonical Correlation Analysis

- All methods discussed assumed single source of data, sometimes we have two (or more) variables (examples)
- Dataset has $\mathcal{X} = \{\boldsymbol{x}^t, \boldsymbol{y}^t\}_{t=1}^N \quad \boldsymbol{x}^t \in \mathcal{R}^d, \quad \boldsymbol{y}^t \in \mathcal{R}^e$
- Define *dxd* covariance matrix **x**: $\boldsymbol{S}_{xx} = Cov(\boldsymbol{x}) = E[(\boldsymbol{x} - \boldsymbol{\mu}_x)^2]$
- Define exe covariance matrix **y**: $\boldsymbol{S}_{yy} = Cov(\boldsymbol{y}) = E[(\boldsymbol{y} - \boldsymbol{\mu}_y)^2]$
- Cross co-variance matrices:

$$\boldsymbol{S}_{xy} = Cov(\boldsymbol{x}, \boldsymbol{y}) = E[(\boldsymbol{x} - \mu_x)(\boldsymbol{y} - \boldsymbol{\mu}_y)] \;\; d \times e \text{ matrix}$$

$$\boldsymbol{S}_{yx} = Cov(\boldsymbol{y}, \boldsymbol{x}) = E[(\boldsymbol{y} - \boldsymbol{\mu}_y)(\boldsymbol{x} - \mu_x)] \;\; e \times d \text{ matrix}$$

Canonical Correlation Analysis

- We seek to find two vectors, ***w*** and ***v***, such that when ***x*** (***y***) is projected along ***w*** (***v***), we have the maximum correlation. So we want to maximize

$$\rho = \mathrm{Corr}(\boldsymbol{w}^T\boldsymbol{x}, \boldsymbol{v}^T\boldsymbol{y}) = \frac{\mathrm{Cov}(\boldsymbol{w}^T\boldsymbol{x}, \boldsymbol{v}^T\boldsymbol{y})}{\sqrt{\mathrm{Var}(\boldsymbol{w}^T\boldsymbol{x})}\sqrt{\mathrm{Var}(\boldsymbol{v}^T\boldsymbol{y})}}$$

$$= \frac{\boldsymbol{w}^T\mathrm{Cov}(\boldsymbol{x}, \boldsymbol{y})\boldsymbol{v}}{\sqrt{\boldsymbol{w}^T\mathrm{Var}(\boldsymbol{x})\boldsymbol{w}}\sqrt{\boldsymbol{v}^T\mathrm{Var}(\boldsymbol{y})\boldsymbol{v}}} = \frac{\boldsymbol{w}^T\mathbf{S}_{xy}\boldsymbol{v}}{\sqrt{\boldsymbol{w}^T\mathbf{S}_{xx}\boldsymbol{w}}\sqrt{\boldsymbol{v}^T\mathbf{S}_{yy}\boldsymbol{v}}}$$

- This is equivalent to :

$$\text{maximize } \boldsymbol{w}^T\boldsymbol{S}_{xy}\boldsymbol{v} \text{ s.t. } \boldsymbol{w}^T\boldsymbol{S}_{xx}\boldsymbol{w} = \boldsymbol{v}^T\boldsymbol{S}_{yy}\boldsymbol{v} = 1$$

Canonical Correlation Analysis

- Writing the above optimization as Lagrangian, take derivatives of ***w*** and ***v*** and equate them to zero. We have:

$$\boldsymbol{w} \text{ is eigenvector of } \boldsymbol{S}_{xx}^{-1}\boldsymbol{S}_{xy}\boldsymbol{S}_{yy}^{-1}\boldsymbol{S}_{yx}$$

$$\boldsymbol{v} \text{ is eigenvector of } \boldsymbol{S}_{yy}^{-1}\boldsymbol{S}_{yx}\boldsymbol{S}_{xx}^{-1}\boldsymbol{S}_{xy}$$

- Since we want to *maximize* the correlation, we choose two eigenvectors with the highest eigenvalues, say $\boldsymbol{w}_1$ and $\boldsymbol{v}_1$ which share the same eigenvalue λ_1.
- Similar to PCA, we can decide how many pairs of eigenvectors ($\boldsymbol{w}_i$, $\boldsymbol{v}_i$), we choose the top *k* using:

$$\frac{\lambda_i}{\sum_{j=1}^{s}\lambda_j} \quad \text{where } s = \min(d, e)$$

Canonical Correlation Analysis

- Let say we chose *k* dimensions. We get the ***canonical variables*** by projecting the training samples along them:

$$a_i^t = \boldsymbol{w}_i^T \boldsymbol{x}^t, \; b_i^t = \boldsymbol{v}_i^T \boldsymbol{y}^t, \quad i = 1, ..., k$$

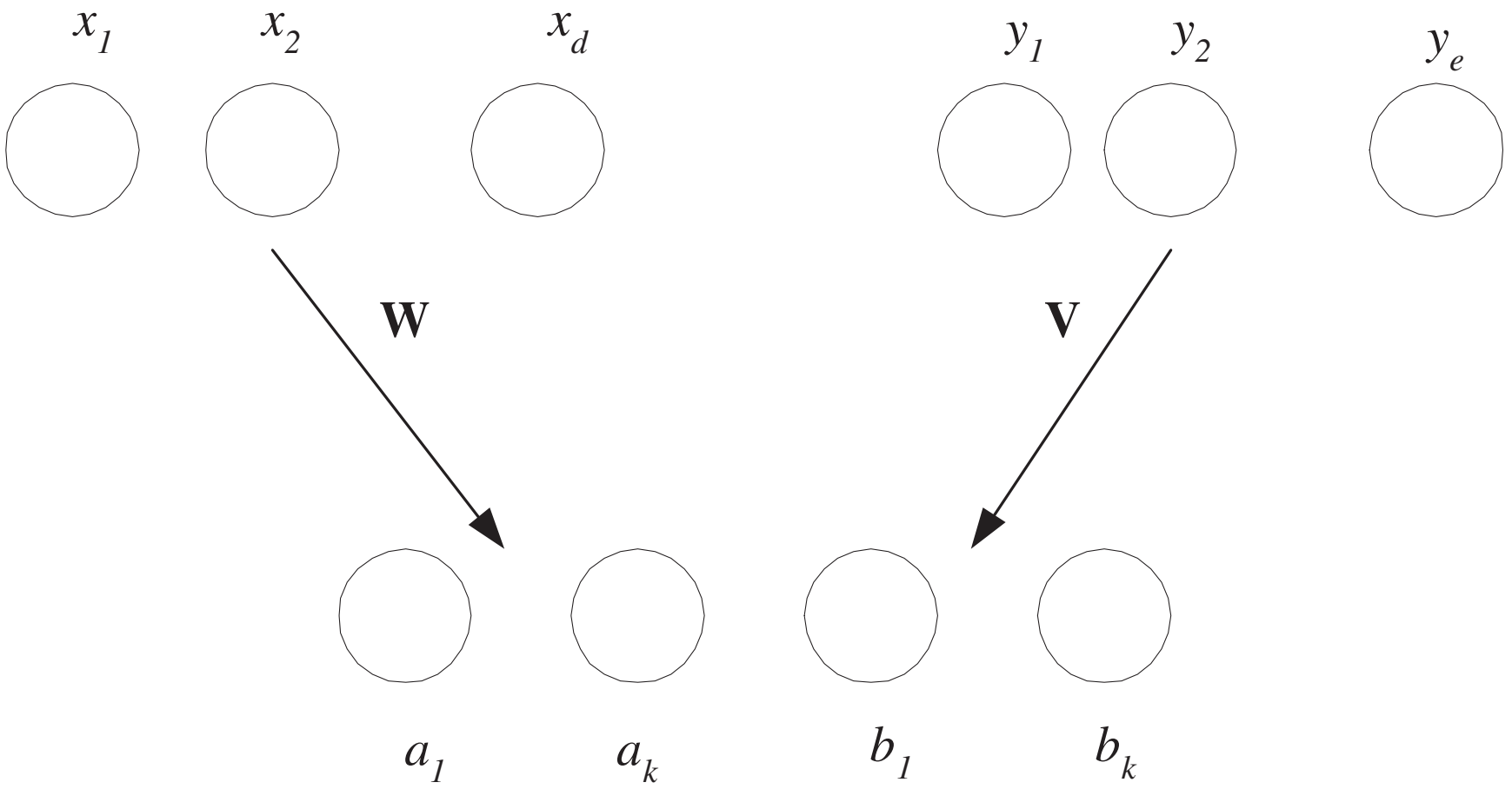
- In matrix form, we have

$$\boldsymbol{a}^t = \boldsymbol{W}^T \boldsymbol{x}^t, \boldsymbol{b}^t = \boldsymbol{V}^T \boldsymbol{y}^t \quad \text{where } \boldsymbol{W} \text{ is } d \times k, \boldsymbol{V} \text{ is } e \times k$$

- The new vector, (a_i, b_i), is the new and lower-dimensional representation
- For CCA to be effective, the two sets of variables need to be ***dependent***.

Canonical Correlation Analysis (CCA)

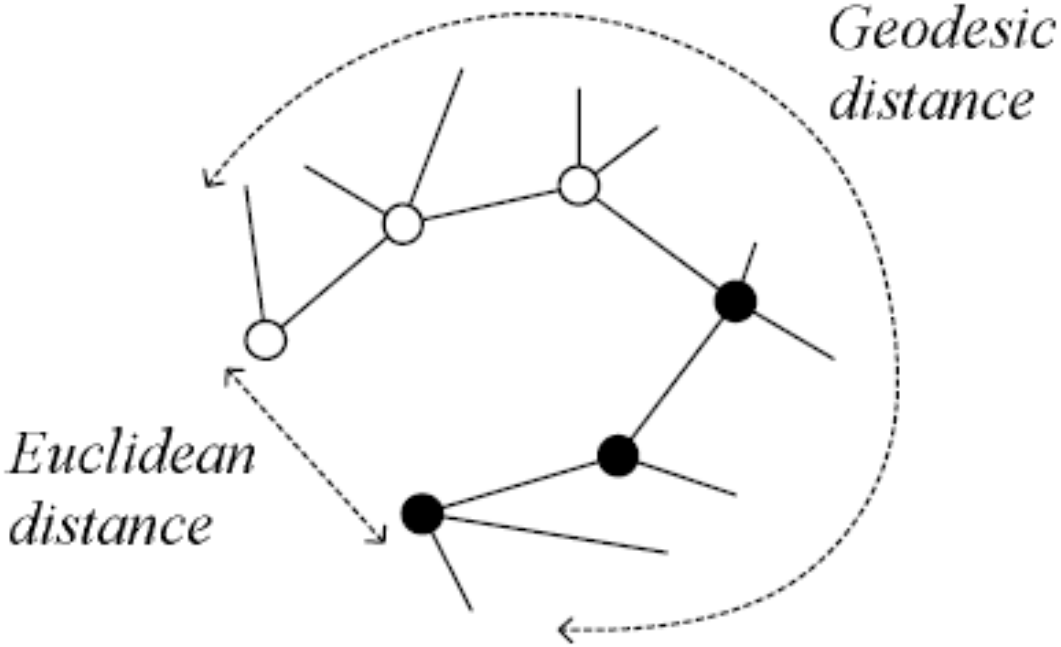
- ***x*** and ***y*** may be two different views or modalities; e.g., image and word tags, and CCA does a joint mapping



Isomap

- **PCA** works when the data lies in a linear space (Euclidean distance)
- **Geodesic distance** is the distance along the **manifold** that the data lies in, as opposed to the Euclidean distance in the input space

sphere and torus are example of manifolds



Isomap

- Instances r and s are connected in the graph if

  $||\boldsymbol{x}^r-\boldsymbol{x}^s||<\varepsilon$ or if $\boldsymbol{x}^s$ is one of the k neighbors of $\boldsymbol{x}^r$

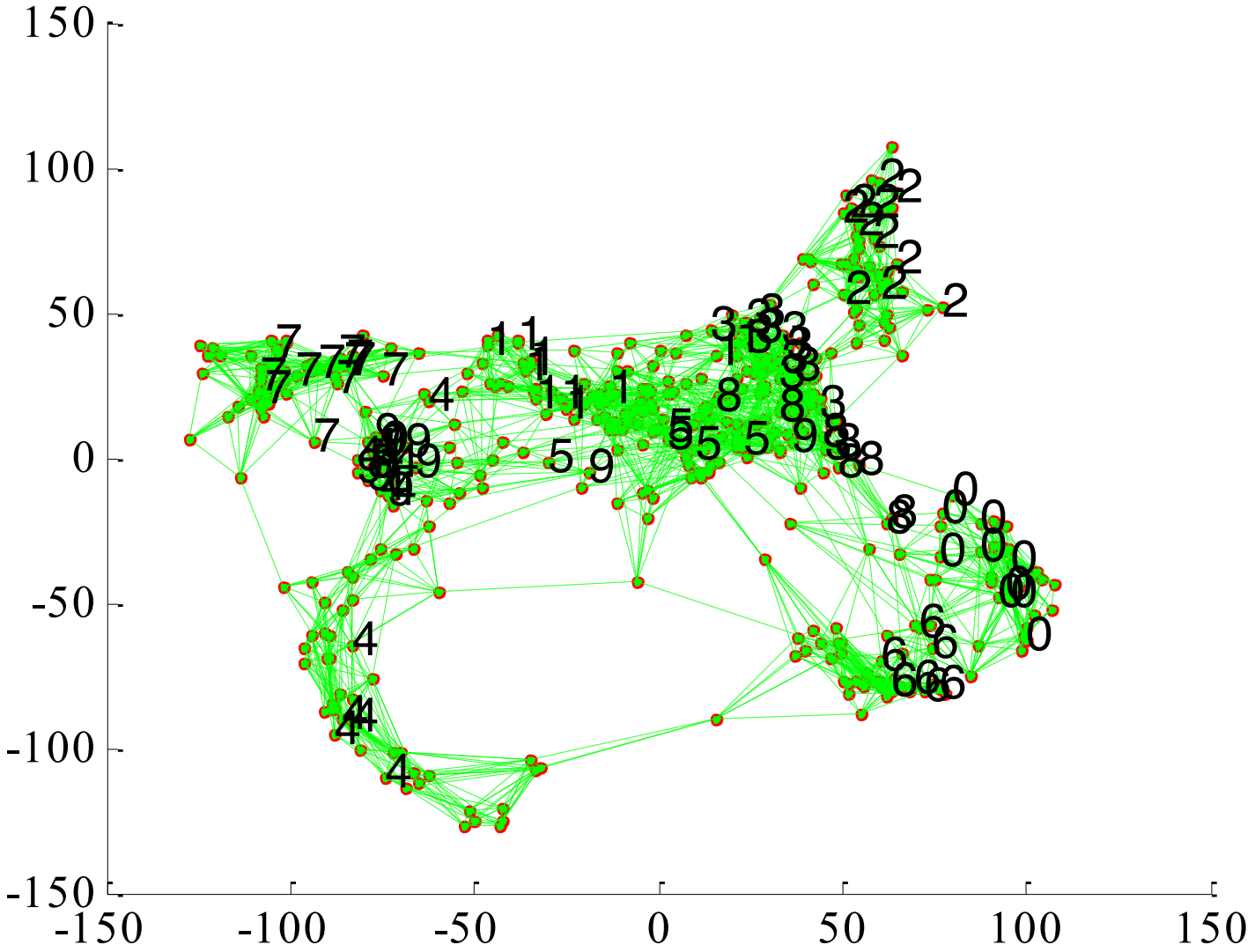
  The edge length is $||\boldsymbol{x}^r-\boldsymbol{x}^s||$
- For two nodes r and s not connected, the distance is equal to the shortest path between them
- Once the NxN distance matrix is thus formed, use MDS to find a lower-dimensional mapping

Isomap

- To compute the geodesic distance on a manifold with *N* points, we define a graph. The distance between two neighboring points are based on Euclidean distance (weight of that edge), distance between two points are the length of the shortest path in the graph.
- Instances r and s are connected in the graph if

  $||\boldsymbol{x}^r - \boldsymbol{x}^s|| < \varepsilon$ or if $\boldsymbol{x}^s$ is one of the *k* neighbors of $\boldsymbol{x}^r$

  The edge length is $||\boldsymbol{x}^r - \boldsymbol{x}^s||$
- For two nodes r and s not connected, the distance is equal to the shortest path between them
- Once the *N*x*N* distance matrix is thus formed, use MDS to find a lower-dimensional mapping



Matlab source from http://web.mit.edu/cocosci/isomap/isomap.html

Locally Linear Embedding (LLE)

- LLE recovers global non-linear structure via locally linear fit
- **Idea**: each local patch of the manifold can be approximated linearly and given enough data, each point can be written as a linear, weighted sum of its neighbors
- Given $\boldsymbol{x}^r$ find its neighbors $\boldsymbol{x}^s_{(r)}$
- First, find the reconstruction weights $\mathbf{W}_{rs}$ that minimize

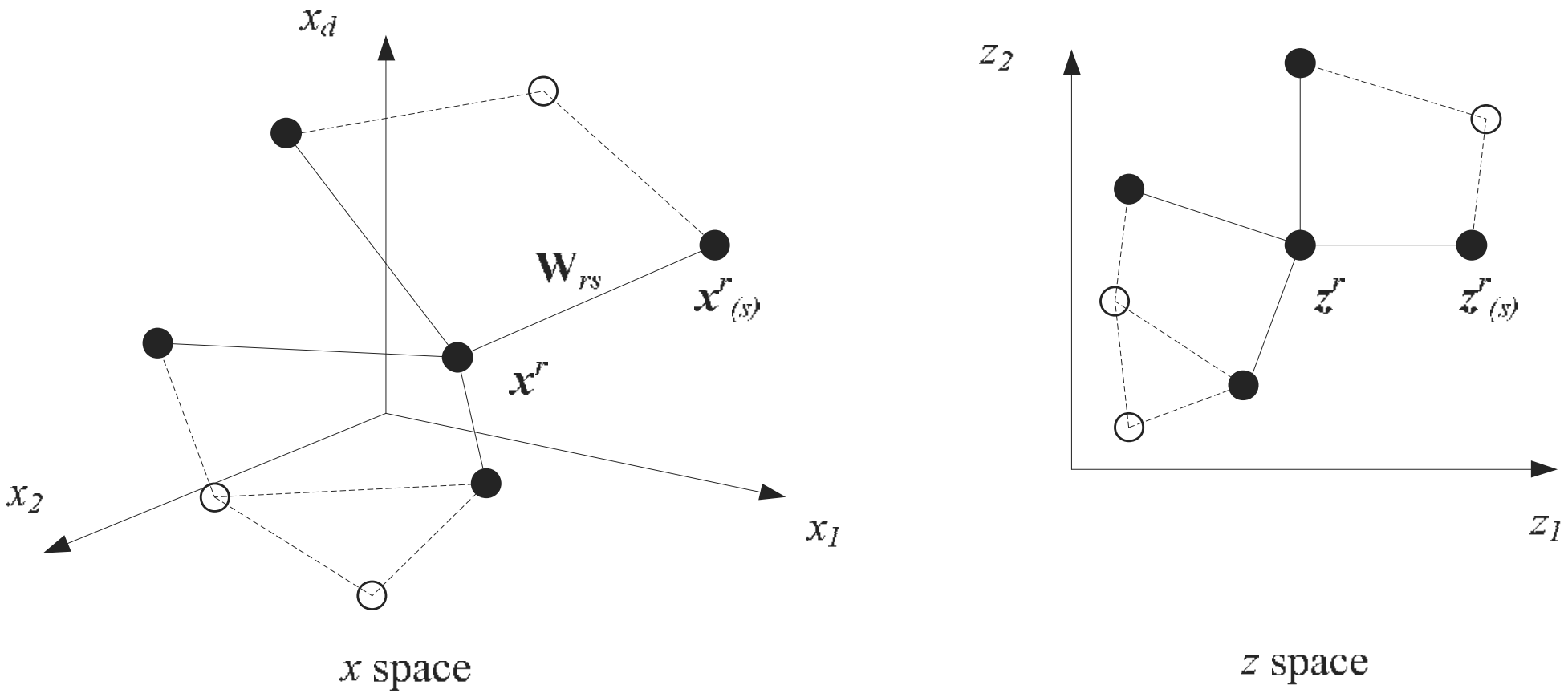
$$\mathcal{E}^w(\mathbf{W}|\mathcal{X}) = \sum_r \|\boldsymbol{x}^r - \sum_s \mathbf{W}_{rs}\boldsymbol{x}^s_{(r)}\|^2$$
$$\text{subject to } \mathbf{W}_{rr} = 0, \forall r \text{ and } \textstyle\sum_s \mathbf{W}_{rs} = 1$$

Locally Linear Embedding (LLE)

- Second, after finding $\mathbf{W}_{rs}$ in step one, find the new coordinates $\boldsymbol{z}^r$ which take whatever values they need but respecting the interpoint constraints given by $\mathbf{W}_{rs}$

$$\mathcal{E}^Z(\mathcal{Z}|\mathbf{W}) = \sum_r \|\boldsymbol{z}^r - \sum_s \mathbf{W}_{rs}\boldsymbol{z}^s\|^2$$

- For detail, read the textbook.

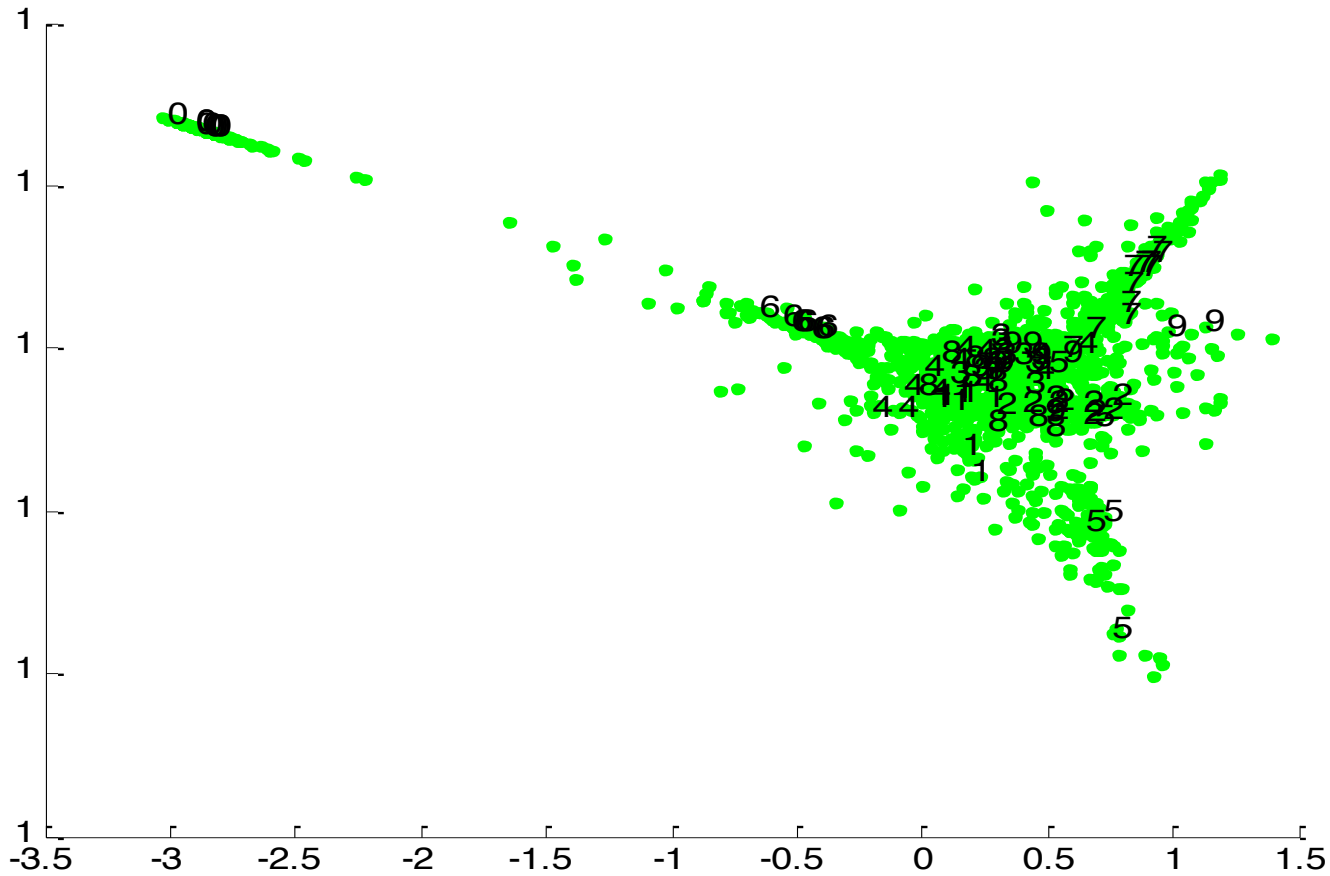


x space

z space

Figure 6.10 Local linear embedding first learns the constraints in the original space and next places the points in the new space respecting those constraints. The constraints are learned using the immediate neighbors (shown with continuous lines) but also propagate to second-order neighbors (shown dashed).

LLE on Optdigits



Matlab source from http://www.cs.toronto.edu/~roweis/lle/code.html

Laplacian Eigenmaps

- Let ***r*** and ***s*** be two instances and B_{rs} is their similarity, we want to find $\boldsymbol{z}^r$ and $\boldsymbol{z}^s$ that

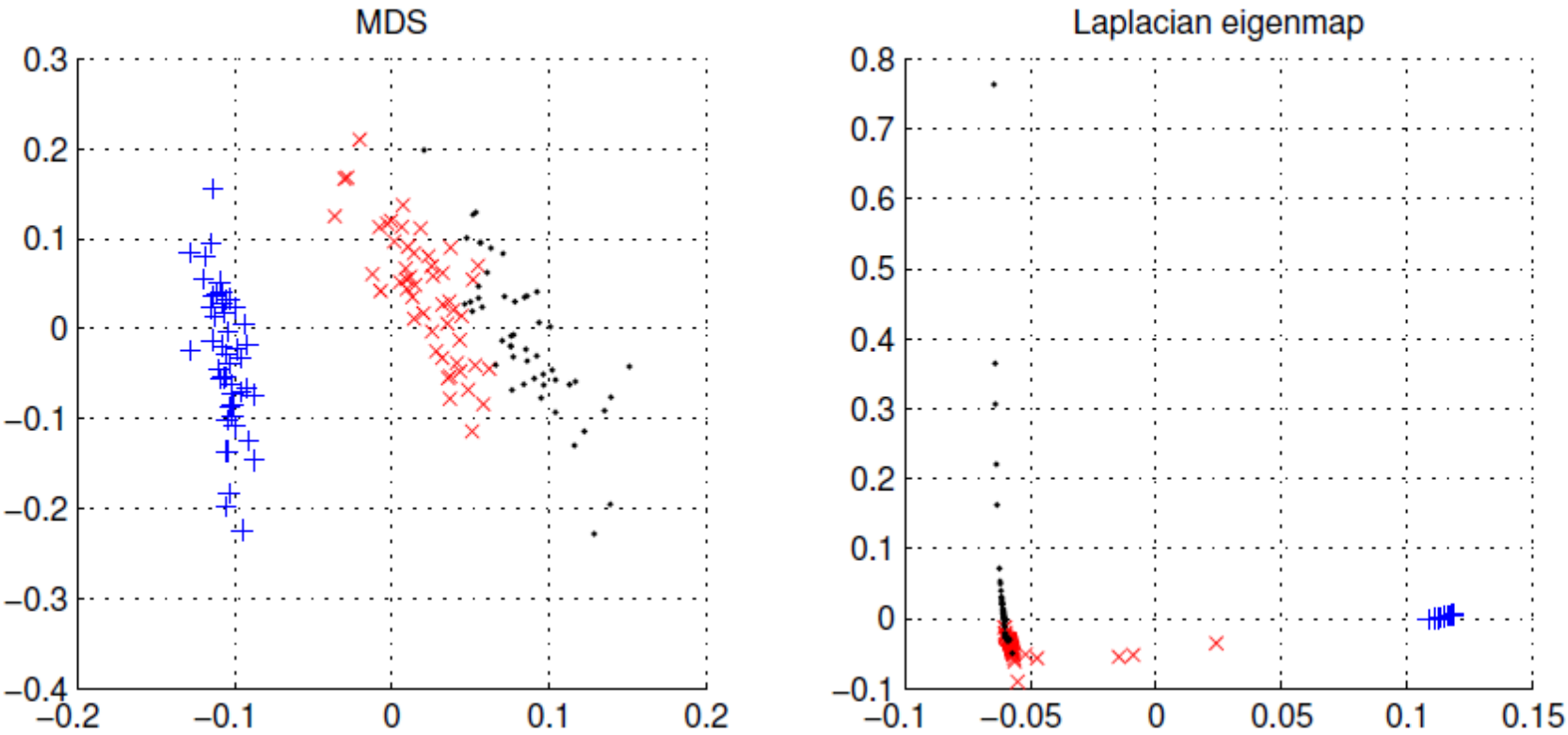
$$\min \sum_{r,s} \|\boldsymbol{z}^r - \boldsymbol{z}^s\|^2 B_{rs}$$

- B_{rs} can be defined in terms of similarity in an original space: 0 if $\boldsymbol{x}^r$ and $\boldsymbol{x}^s$ are too far apart, otherwise

$$B_{rs} = \exp\left[-\frac{\|\boldsymbol{x}^r - \boldsymbol{x}^s\|^2}{2\sigma^2}\right]$$

- Defines a graph Laplacian, and feature embedding returns $\boldsymbol{z}^r$

Laplacian Eigenmaps on Iris



Spectral clustering (chapter 7)