Dimensionality Reduction & Visualization

Ishwar K Sethi

Dimensionality Reduction

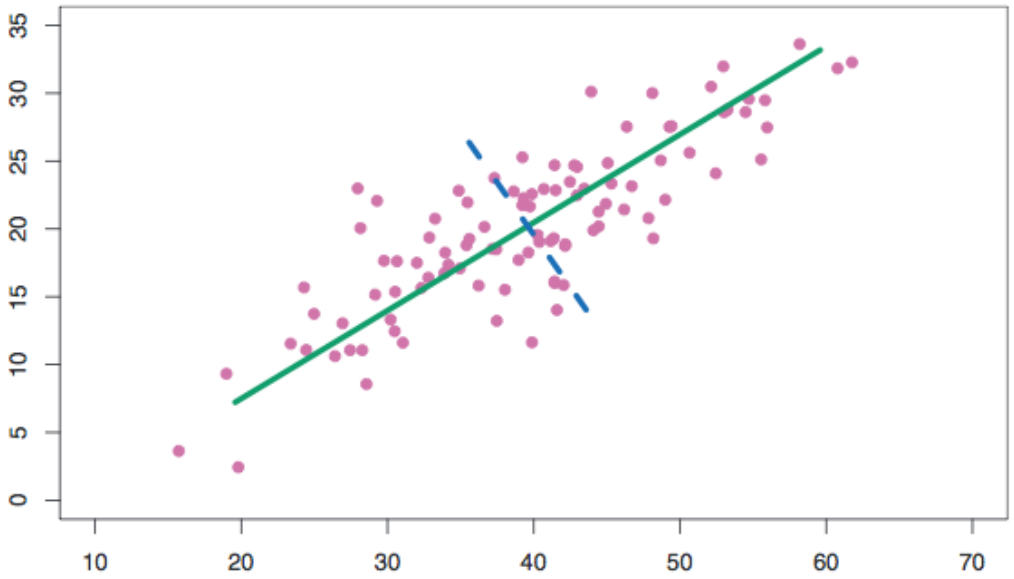
- Purpose:
  - Avoid curse of dimensionality
  - Reduce amount of time and memory required by data mining algorithms
  - Allow data to be more easily visualized
  - May help to eliminate irrelevant features or reduce noise
- Techniques
  - Several linear (PCA/SVD) or nonlinear (t-SNE)
  - Supervised and unsupervised

Principal Component Analysis (PCA)

Dimensionality Reduction: PCA

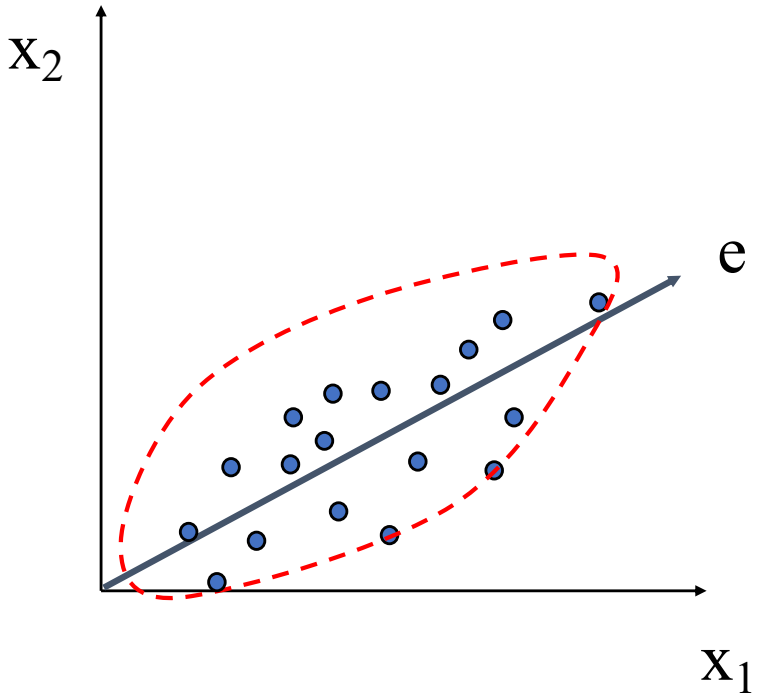
- Goal is to find a projection that captures the largest amount of variation in data. Such a projection of the data also best represents (approximates) the data in the least square sense

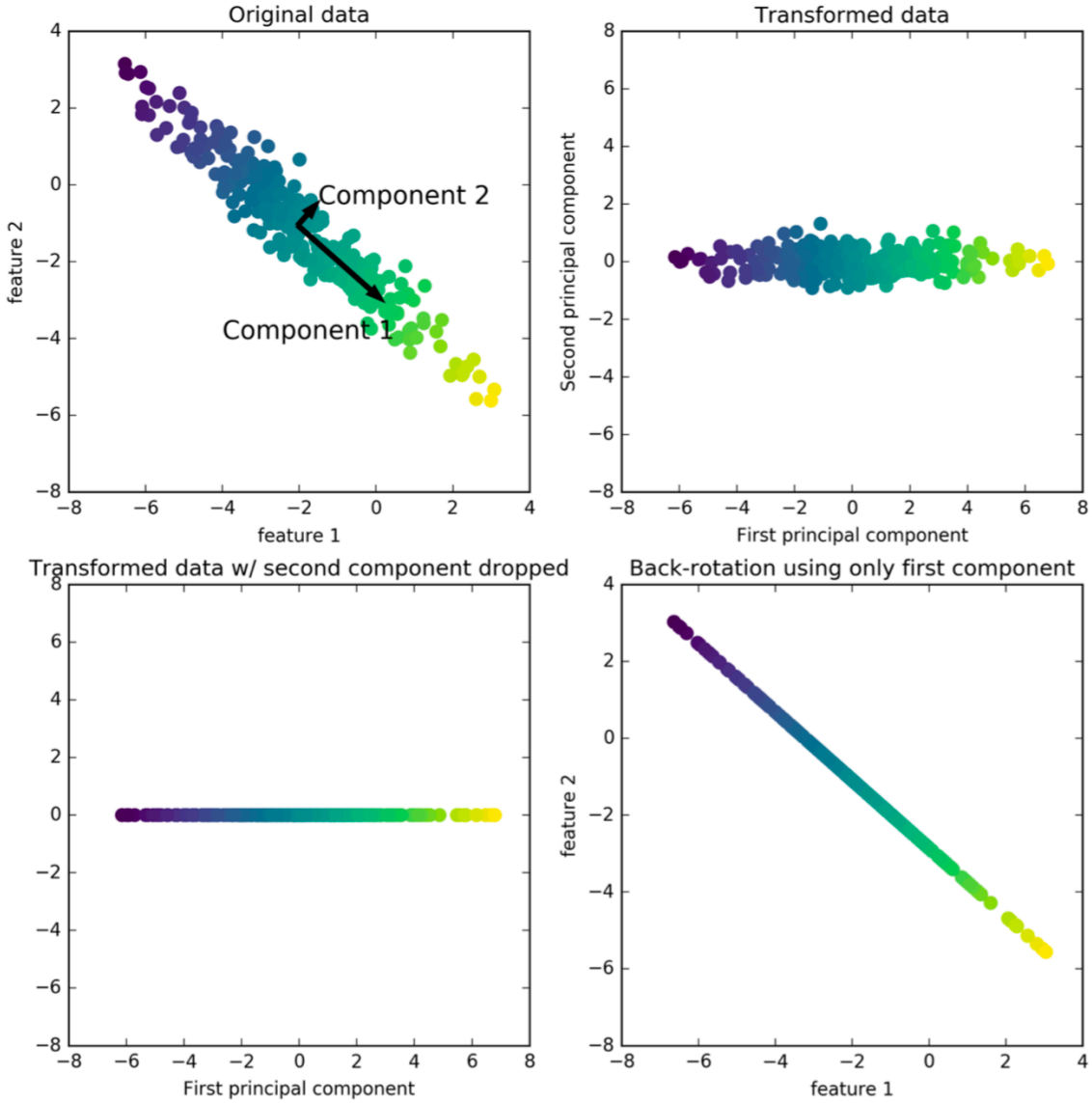
$$\min \; ||\boldsymbol{x} - \hat{\boldsymbol{x}}||^2$$



PCA Steps

- Find the eigenvectors of the sample covariance matrix using the given data
- The eigenvectors define the new space




Original data
Component 2
Component 1
feature 2
feature 1
Transformed data
Second principal component
First principal component
Transformed data w/ second component dropped
First principal component
Back-rotation using only first component
feature 2
feature 1

PCA Basics

- PCA takes *n* variables $x_1, x_2, \ldots, x_n$ and finds linear combinations of these variables to produce a new set of variables $y_1, y_2, \ldots, y_n$ that are uncorrelated. The transformed variables are indexed or ordered so that y_1 shows the largest amount of variation, y_2 has the second largest amount of variation, and so on. In other words,

$$\text{var}(y_1) \geq \text{var}(y_2) \geq \cdots \text{var}(y_n)$$

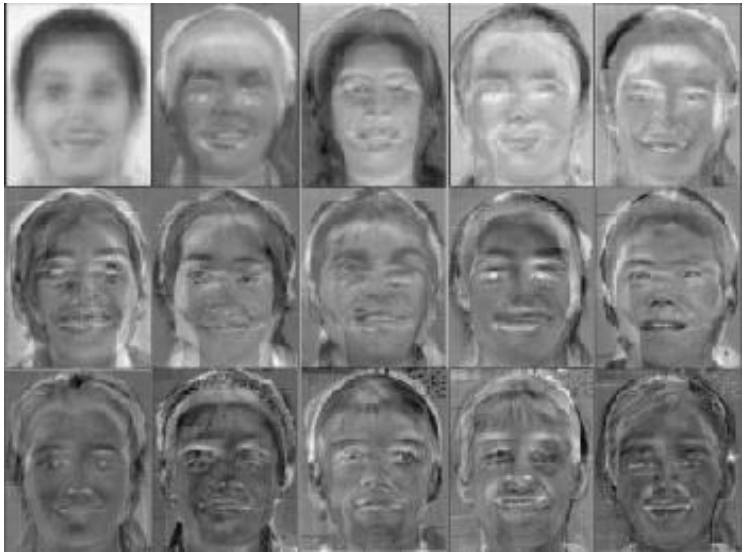
  where var(.) stands for variance.

- The expectation while performing PCA is that the variances of most of the transformed variables will be negligibly low. In such cases, the original data set can be described by few transformed variables thus achieving dimensionality reduction. The PCA is a very popular technique with many numerous uses.

- PCA is also known as *Karhunen-Loeve transform*, *Hotelling transform*, or the *eigenvector analysis*.

Eigenfaces: A PCA Application

Through PCA, each face on left can be represented as a weighted linear sum of eigenfaces shown below



PCA Math

- We start with *n*-dimensional *N* data vectors

  $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N$

- We use the following linear transformation to obtain principle components for

$$\boldsymbol{y}_i = \boldsymbol{A}(\boldsymbol{x}_i - \boldsymbol{M}_x) \text{ for } i = 1, N$$

where $\boldsymbol{m}_x = \frac{1}{N} \sum_{k=1}^{N} \boldsymbol{x}_k,$ and $\boldsymbol{A}$ is a matrix whose rows are formed from the eigenvectors of the sample covariance matrix.

PCA Math

- Sample covariance matrix

$$\boldsymbol{C}_x = \frac{1}{N-1}\sum_{k=1}^{N}(\boldsymbol{x}_k - \boldsymbol{m}_x)(\boldsymbol{x}_k - \boldsymbol{m}_x)^t$$

- The rows of the transformation matrix $\boldsymbol{A}$ are ordered so that the first row of the matrix is the eigen vector corresponding to the largest eigen value of the sample covariance matrix, and the second row is formed by the eigen vector corresponding to the second largest eigen value and so on.

$$c_{pp} = \sigma_{pp}^2 = \frac{\sum_i (x_{p_i} - m_{x_p})(x_{p_i} - m_{x_p})}{N-1}$$

Covariance matrix' diagonal terms

$$c_{pq} = \sigma_p \sigma_q = \frac{\sum_i (x_{p_i} - m_{x_p})(x_{q_i} - m_{x_q})}{N-1}$$

Non-diagonal terms

PCA Math

- What is the mean of the transformed vectors ($\boldsymbol{m}_y$)?
- What about the covariance matrix of the transformed vectors, i.e.

$$\boldsymbol{C}_y = \boldsymbol{A}\boldsymbol{C}_{\boldsymbol{x}}\boldsymbol{A}^{\boldsymbol{t}}?$$

$$= \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_n \end{bmatrix}$$

PCA Math

- Suppose we want to reduce the number of features from *n* to *p, p* < *n.* This is done by keeping only the first *p* rows of the transformation matrix.
- Reducing the number of features will always lead to approximation of the original data if we were to go back to *n* features. The mean square error of approximation is given by the following formula

$$e_{mse} = \sum_{j=p+1}^{n} \lambda_j$$

- Another way to represent approximation is by the following formula where *P* accounts for the data variability in first *p* eigen values

$$P = \frac{\sum_{j=1}^{p} \lambda_j}{\sum_{j=1}^{n} \lambda_j}$$

The best results from PCA are obtained in situations where the original variables are highly correlated. In such instances, it is common to find two or three principal components that account for most of the variability in data.

PCA Example

7	4	3
4	1	8
6	3	5
8	6	1
8	5	7
7	2	9
5	3	3
9	5	8
7	4	5
8	2	2

10 observation vectors; 3 features/vector

6.9	3.5	5.1

Calculate mean vector

2.32222222	1.61111111	-0.43333333
1.61111111	2.5	-1.27777777
-0.43333333	-1.27777777	7.87777777

Calculate sample covariance matrix

PCA Numerical Example

8.274	3.676	0.749

Calculated eigenvalues

0.13757	-0.699	-0.701
0.25046	-0.6608	0.707
-0.9583	-0.273	0.084

Calculated eigenvectors (in columns)

2.151417	0.173
-3.804173	2.8874
-0.153213	0.9868
4.706507	-1.3016
-1.293753	-2.2788
-4.099303	-0.1434
1.625817	2.2318
-2.114483	-3.2508
0.234817	-0.373
2.746367	1.0686

Two-dimensional representation of the original data using the first two eigenvectors

PCA Numerical Example

7.08	3.92	2.99
4.36	0.64	7.96
6.19	2.81	4.98
8.46	5.54	0.95
8.31	4.68	6.96
6.44	2.57	9.07
5.56	2.43	2.93
8.88	5.12	8.01
7.19	3.81	4.98
6.53	3.48	2.18

Reconstructed data

7	4	3
4	1	8
6	3	5
8	6	1
8	5	7
7	2	9
5	3	3
9	5	8
7	4	5
8	2	2

Original data

Mean Square Error = 0.674
(9/10)*(Eigenvalue#3)=0.674

PCA Example

```
import numpy as np
from numpy import linalg as LA
import matplotlib.pyplot as plt
```

```
X = np.array([[7,4,3],[4,1,8],[6,3,5],[8,6,1],[8,5,7],[7,2,9],[5,3,3],[9,5,8],[7,4,5], [8,2,2]])
Xmean = np.mean(X,0)
print(Xmean)
```

```
[6.9 3.5 5.1]
```

```
C = np.cov(X.T)
print(C)
```

```
[[ 2.32222222  1.61111111 -0.43333333]
 [ 1.61111111  2.5        -1.27777778]
 [-0.43333333 -1.27777778  7.87777778]]
```

https://iksinc.online/2018/08/21/principal-component-analysis-pca-explained-with-examples/

PCA Example

```
w, v = LA.eig(C)# Get the eigen values and eigen vectors
print(w)
print(v)
```

```
[0.74992815 3.67612927 8.27394258]
[[-0.70172743  0.69903712 -0.1375708 ]
 [ 0.70745703  0.66088917 -0.25045969]
 [ 0.08416157  0.27307986  0.95830278]]
```

```
A = np.array([v[:,2],v[:,1]])# Form the transformation matrix using eigen vectors corresponding
# to the top two eigen values
print(A)
```

```
[[-0.1375708  -0.25045969  0.95830278]
 [ 0.69903712  0.66088917  0.27307986]]
```

PCA Example

```
Y = np.matmul(A,(X-Xmean).T)# Apply transformation to obtain new data representation in 2-D
print(Y)
```

```
[[-2.15142276  3.80418259  0.15321328 -4.7065185   1.29375788  4.0993133
  -1.62582148  2.11448986 -0.2348172  -2.74637697]
 [-0.17311941 -2.88749898 -0.98688598  1.30153634  2.27912632  0.1435814
  -2.23208282  3.2512433   0.37304031 -1.06894049]]
```

```
xhat = np.matmul(A.T,Y).T + Xmean # Recover the original data
```

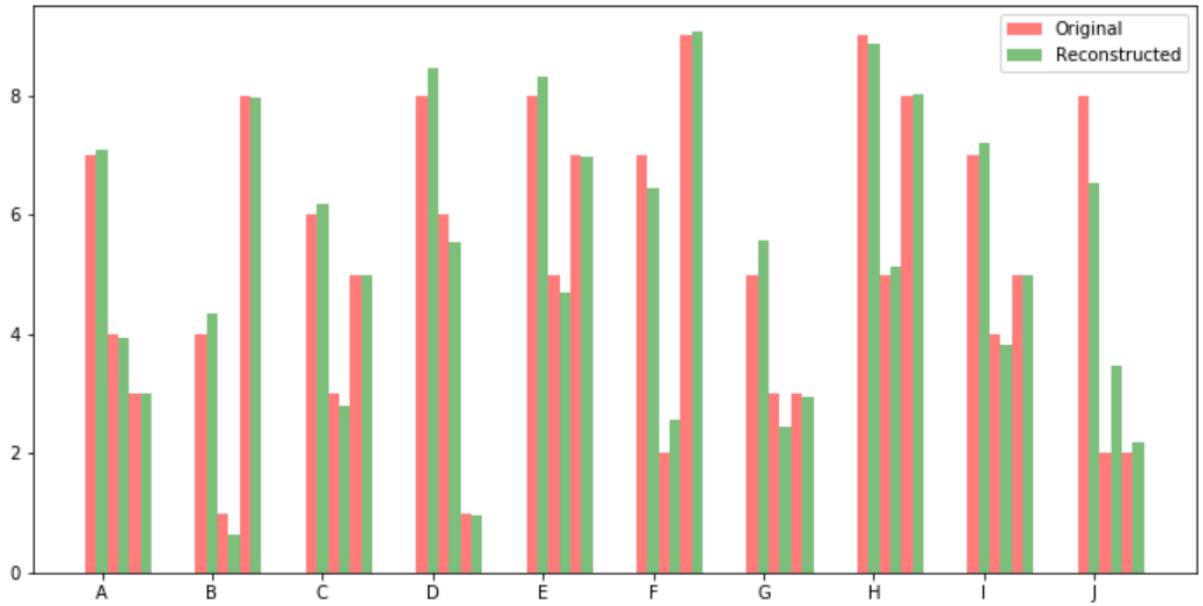
```
print(xhat)
```

```
[[7.07495606 3.92443193 2.99101016]
 [4.35818659 0.63888882 7.95704095]
 [6.18905239 2.80940399 4.97732603]
 [8.45730172 5.53896441 0.94515359]
 [8.31521059 4.68221571 6.96219527]
 [6.43642293 2.56817868 9.06759253]
 [5.56335682 2.43204338 2.93243389]
 [8.88184769 5.11911702 8.01417058]
 [7.19307301 3.80535054 4.97684382]
 [6.53059219 3.48140552 2.17623319]]
```

```
mse = np.sum((X - xhat)**2)/10
print(mse)
```

```
0.6749353375153229
```

```
#Make a bar chart of original and recovered data vectors
plt.figure(figsize=(12,6))
bw = 0.10
r1 = np.arange(len(xhat))
r2 = [x + bw for x in r1]
r3 = [x + bw for x in r2]
r4 = [x + bw for x in r3]
r5 = [x + bw for x in r4]
r6 = [x + bw for x in r5]
plt.bar(r1, X[:,0], bw, alpha=0.5, color='r',label = 'Original')
plt.bar(r2, xhat[:,0], bw, alpha=0.5, color='g', label = 'Reconstructed')
plt.bar(r3, X[:,1], bw, alpha=0.5, color='r')
plt.bar(r4, xhat[:,1], bw, alpha=0.5, color='g')
plt.bar(r5, X[:,2], bw, alpha=0.5, color='r')
plt.bar(r6, xhat[:,2], bw, alpha=0.5, color='g')
plt.xticks([r + bw for r in range(len(xhat))], ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H','I','J'])
plt.legend()
plt.show()
```



PCA using sklearn kit

```
from sklearn import decomposition
pca = decomposition.PCA(n_components=2)
pca.fit(X)
Y = pca.transform(X)
print(Y)
```

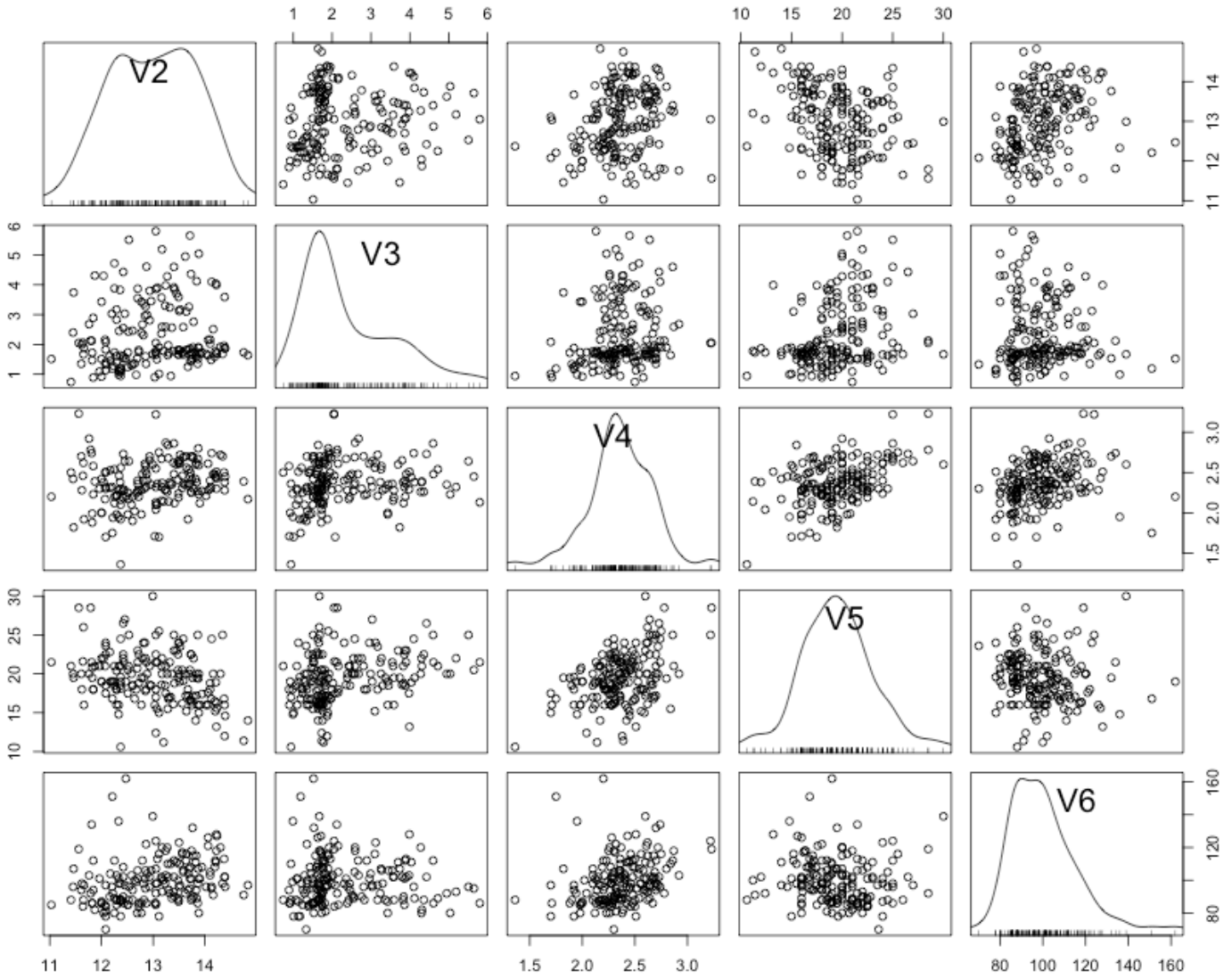
```
[[ 2.15142276 -0.17311941]
 [-3.80418259 -2.88749898]
 [-0.15321328 -0.98688598]
 [ 4.7065185   1.30153634]
 [-1.29375788  2.27912632]
 [-4.0993133   0.1435814 ]
 [ 1.62582148 -2.23208282]
 [-2.11448986  3.2512433 ]
 [ 0.2348172   0.37304031]
 [ 2.74637697 -1.06894049]]
```

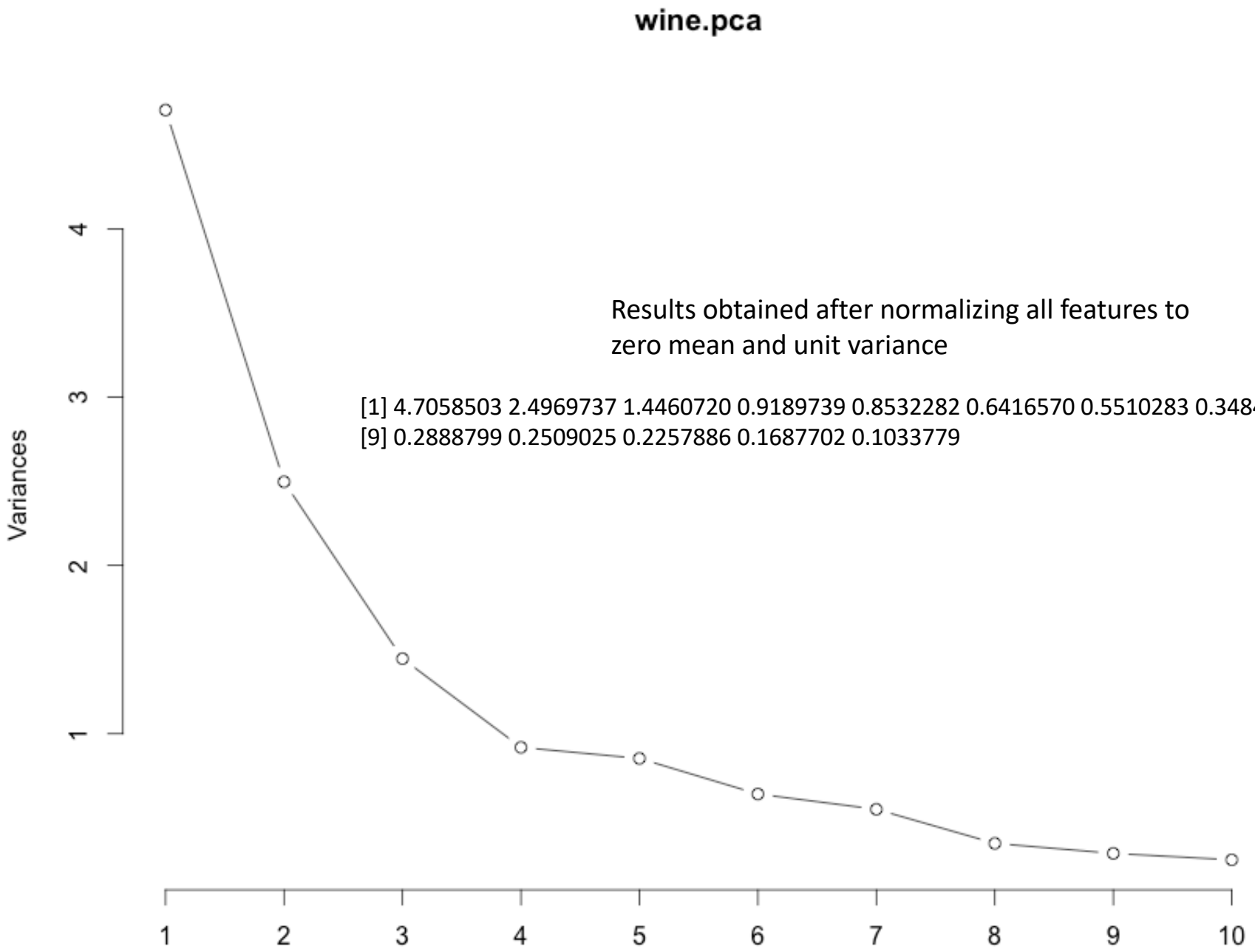
```
xhat = pca.inverse_transform(Y)
mse = np.sum((X - xhat)**2)/10
print(mse)
```

```
0.6749353375153226
```

PCA Illustration

Wine dataset: 3 categories, 13 features, and 178 examples



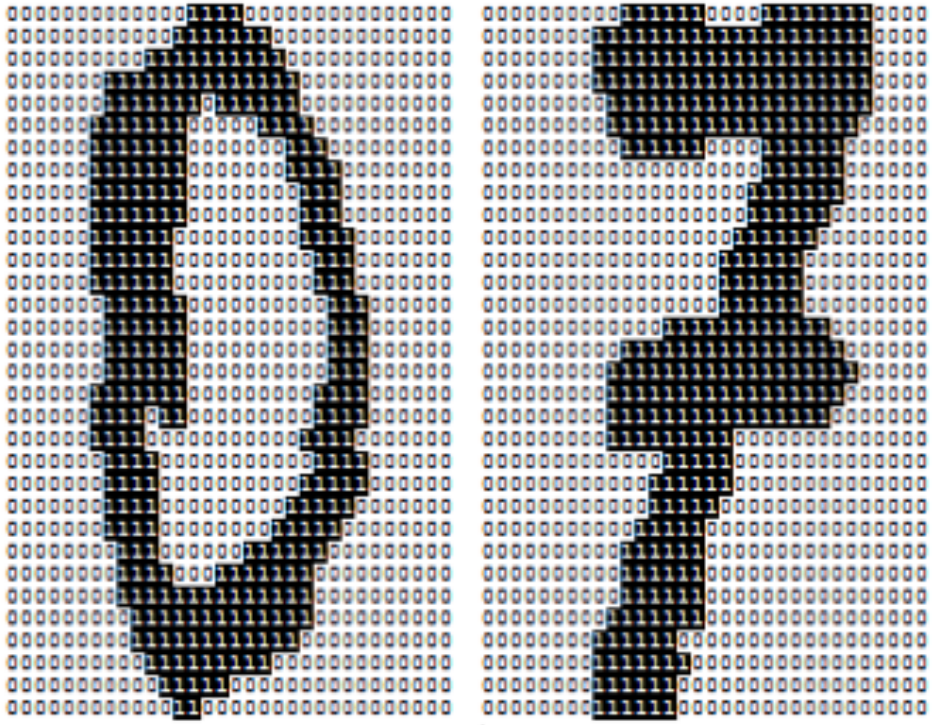


Results obtained after normalizing all features to zero mean and unit variance

[1] 4.7058503 2.4969737 1.4460720 0.9189739 0.8532282 0.6416570 0.5510283 0.3484974
[9] 0.2888799 0.2509025 0.2257886 0.1687702 0.1033779

Plot of Wine data using the first two eigenvectors

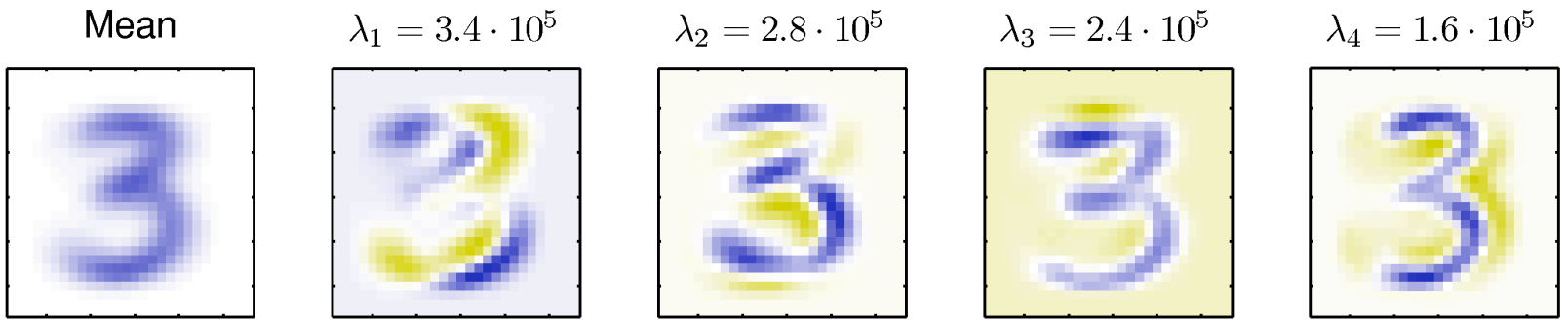
PCA Results for Optdigits Dataset



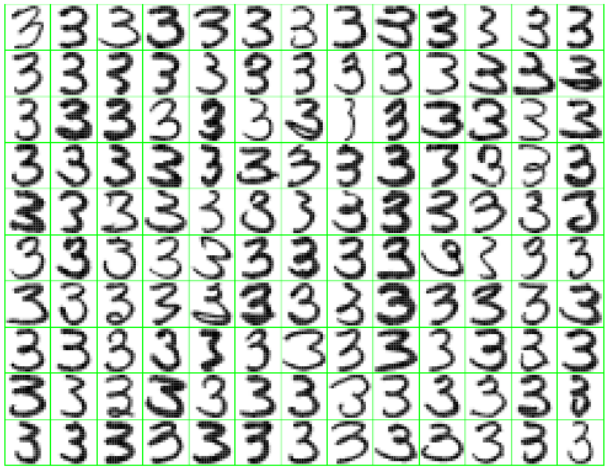
Sample digits extracted from the raw Optdigits dataset.

Dataset of binary images of handwritten digits

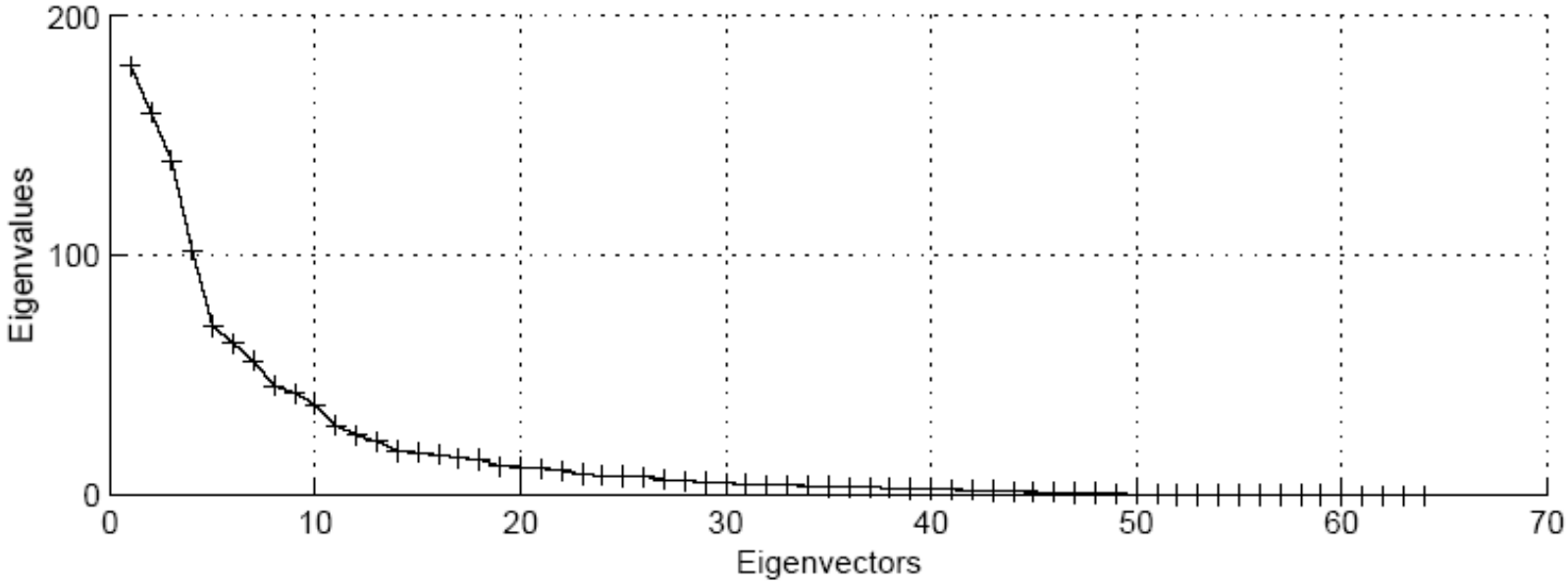
PCA Illustration



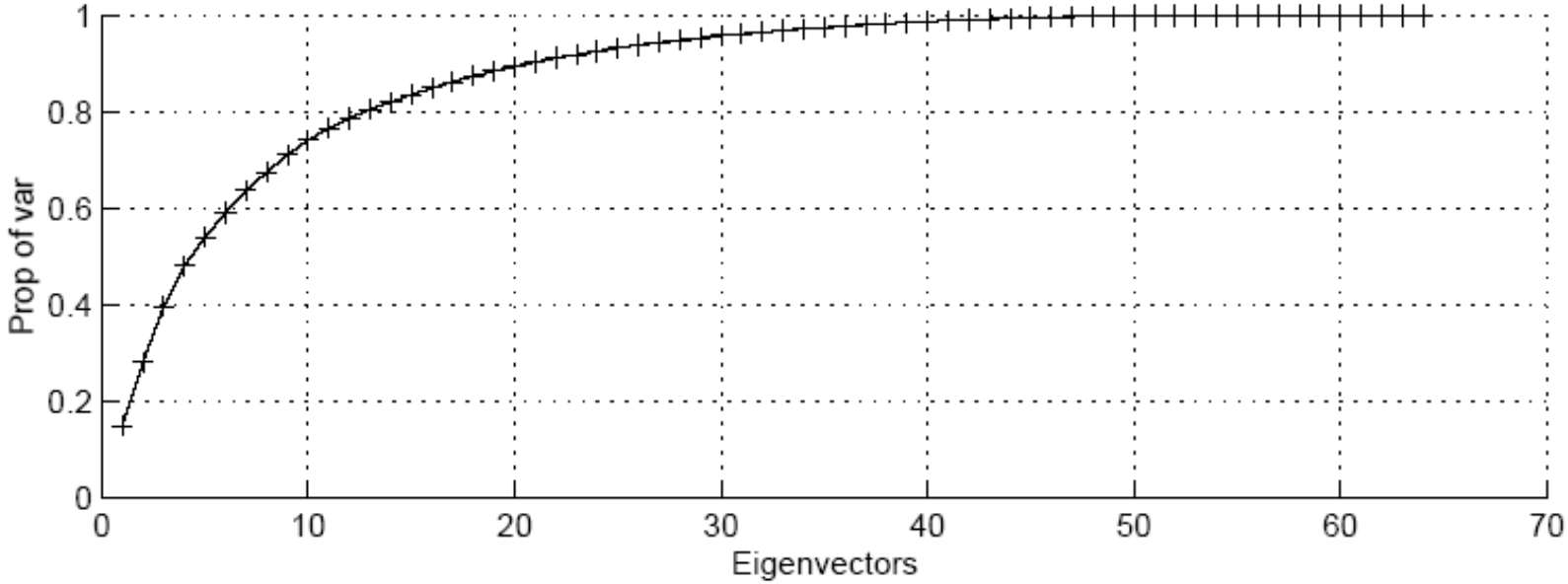
The mean vector and the first four eigenvectors for a collection of digit three images. Blue corresponds to positive values; white is zero and yellow corresponds to negative values.

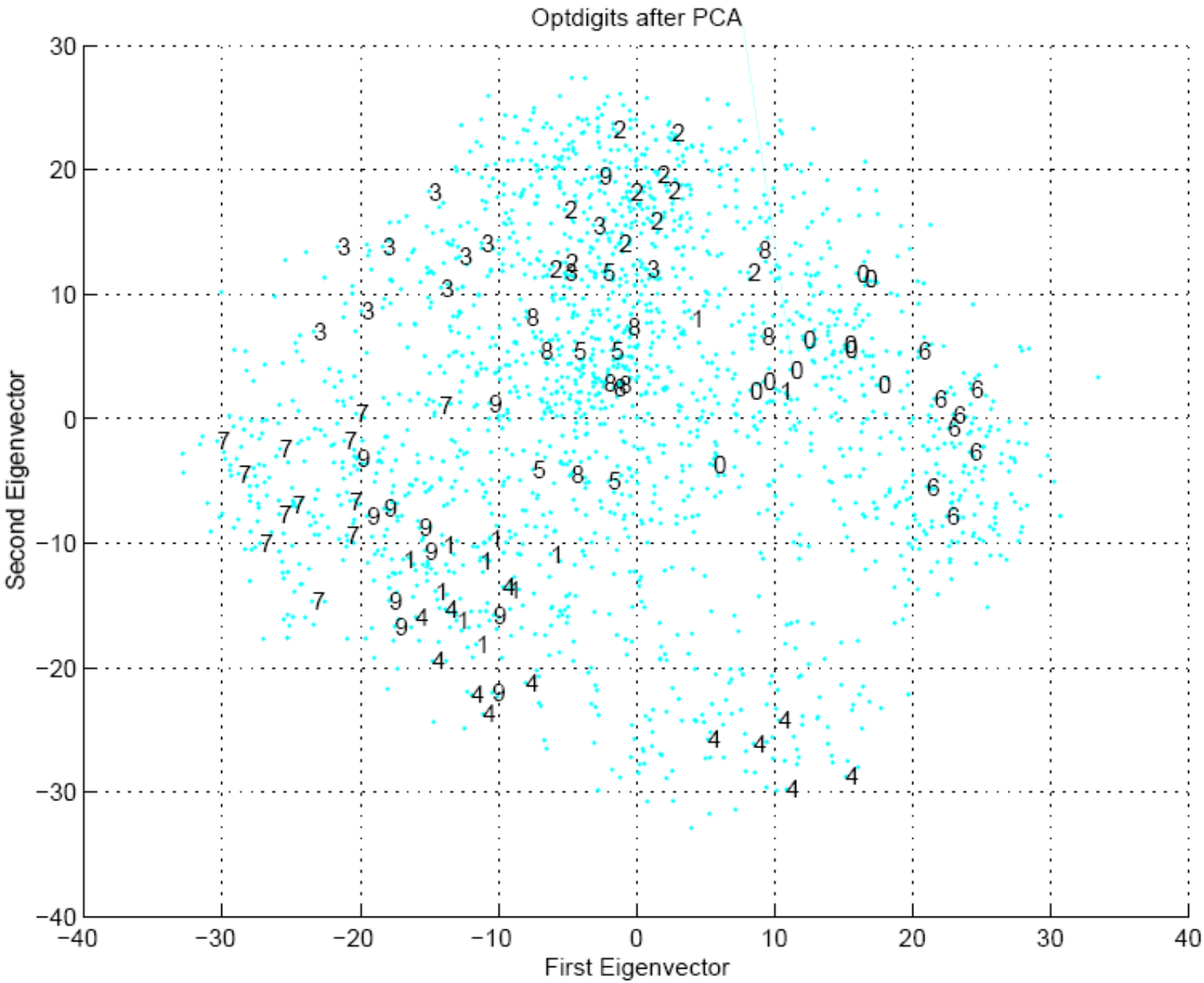


PCA Results for Optdigits Dataset



(b) Proportion of variance explained




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

How to Use PCA When N<d

- In many situations there are fewer data points than the dimensionality of the feature space (For example: images, bioinformatics etc.)
- With N<d, there is a linear subspace of at most N-1 dimensions. Thus, d-N+1 eigenvalues are zero
- In such cases, SVD (Singular Value Decomposition) approach is used for finding eigenvalues and eigenvectors.

Singular Value Decomposition (SVD)

- A technique for handling matrices (sets of equations) that do not have an inverse. This includes square matrices whose determinant is zero and all rectangular matrices.
- Handy mathematical technique that has application to many problems
- According to SVD, an arbitrary matrix **F** of size m x n can be expressed as

$$\mathbf{U}^t\mathbf{F}\mathbf{V} = \mathbf{\Lambda}^{1/2},$$

where **U** and **V** are orthogonal matrices of size m x m and n x n, respectively. The matrix $\mathbf{\Lambda}^{1/2}$ is a mxn diagonal matrix as shown below

$$\Lambda^{1/2} = \begin{bmatrix} \lambda_1^{1/2} & & & \vdots & \\ & \ddots & & \vdots & 0 \\ & & \lambda_r^{1/2} & \vdots & \\ \cdots & \cdots & \cdots & \vdots & \cdots \\ & 0 & & \vdots & 0 \end{bmatrix}$$

Singular Value Decomposition (SVD)

- We can also write $\mathbf{F} = \mathbf{U}\,\mathbf{\Lambda}^{1/2}\,\mathbf{V}^t$, since **U** and **V** are orthogonal matrices.
- The columns of the matrix **U** are composed of the eigenvectors of the symmetric matrix $\mathbf{FF}^t$, and the columns of the matrix **V** are the eigenvectors of the symmetric matrix $\mathbf{F}^t\mathbf{F}$.
- In terms of eigenvectors, it is possible to express matrix **F** as follows:

$$\mathbf{F} = \sum_{j=1}^{r} \lambda_j^{1/2} \mathbf{u}_j \mathbf{v}_j^t$$

- The outer products of the eigenvectors above form a set of unit rank matrices each of which is scaled by a corresponding singular value of **F**.

SVD

- The diagonal elements of the matrix $\mathbf{\Lambda}^{1/2}$ are called the singular values of **A**
- If **A** is singular, some of the diagonal elements will be 0
- In general *rank*(**A**) = number of nonzero *diagonal elements of* $\mathbf{\Lambda}^{1/2}$

Example

Let us find SVD for matrix $\mathbf{F} = \begin{bmatrix} 6 & 6 \\ 0 & 1 \\ 4 & 0 \\ 0 & 6 \end{bmatrix}$

Step 1: Compute **V**.

$\mathbf{F}^t\mathbf{F} = \begin{bmatrix} 52 & 36 \\ 36 & 73 \end{bmatrix}$

The eigenvalues of the above matrix are 100 and 25, respectively.
The corresponding eigenvectors are $[0.6\ 0.8]^t$ and $[0.8\ -0.6]^t$.

Step 2: Compute **U**.

$\mathbf{F}\mathbf{F}^t = \begin{bmatrix} 72 & 6 & 24 & 36 \\ 6 & 1 & 0 & 6 \\ 24 & 0 & 16 & 0 \\ 36 & 6 & 0 & 36 \end{bmatrix}$

This matrix has only two nonzero eigenvalues (same as above) and the corresponding eigenvectors are $[0.84\ 0.08\ 0.24\ 0.48]^t$ and $[0.24\ -0.12\ 0.64\ -0.72]^t$.

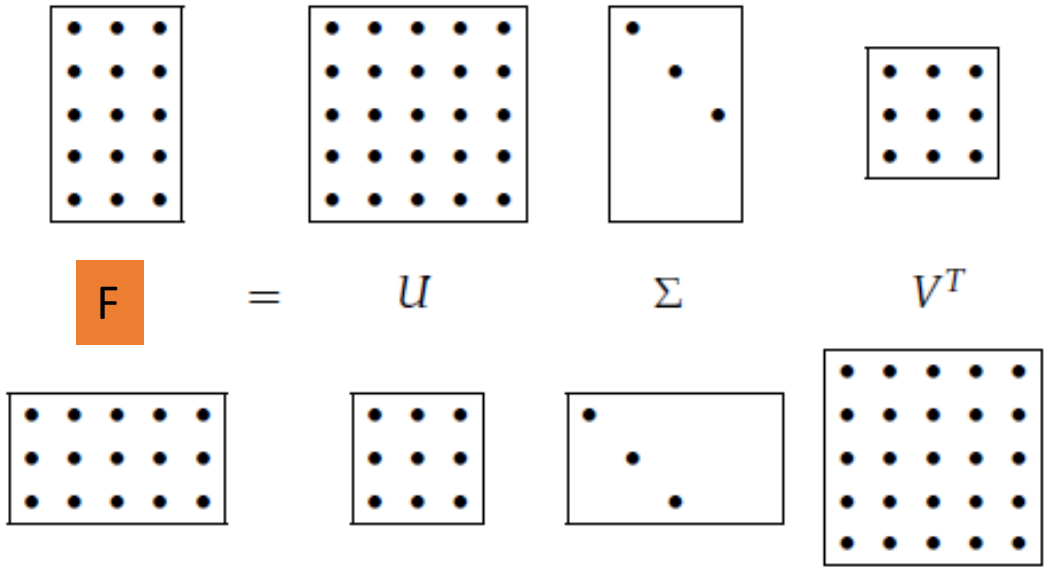
Example (Cntnd)

Step 3: Express **F** in SVD form:

$$\mathbf{F} = (100)^{1/2}[0.84\ 0.08\ 0.24\ 0.48]^t[0.6\ 0.8] + (25)^{1/2}[0.24\ -0.12\ 0.64\ -0.72]^t[0.8\ -0.6]$$

$$\mathbf{F} = \begin{bmatrix} 5.04 & 6.72 \\ 0.48 & 0.64 \\ 1.44 & 1.92 \\ 2.88 & 3.84 \end{bmatrix} + \begin{bmatrix} 0.96 & -0.72 \\ -0.48 & 0.36 \\ 2.56 & -1.92 \\ -2.88 & 2.16 \end{bmatrix} = \begin{bmatrix} 6 & 6 \\ 0 & 1 \\ 4 & 0 \\ 0 & 6 \end{bmatrix}$$

SVD Illustration



The top half illustrates the case when matrix F has more rows than columns. The lower half shows the opposite case.

SVD and Matrix Similarity

- One common definition for the norm of a matrix is the Frobenius norm:

$$\|F\|_{\text{FNorm}} = \sum_i \sum_j f_{ij}^2$$

- Frobenius norm can be computed from SVD

$$\|F\|_{\text{FNorm}} = \sum_i \lambda_i^2$$

- So changes to a matrix can be evaluated by looking at changes to singular values

Approximation using SVD

- An approximation of matrix **F** can be obtained by expressing it as the sum of *k* m x n rank-one matrices:

$$\hat{\mathbf{F}} = \sum_{j=1}^{k} \lambda_j^{1/2} \mathbf{u}_j \mathbf{v}_j^t, k \le r$$

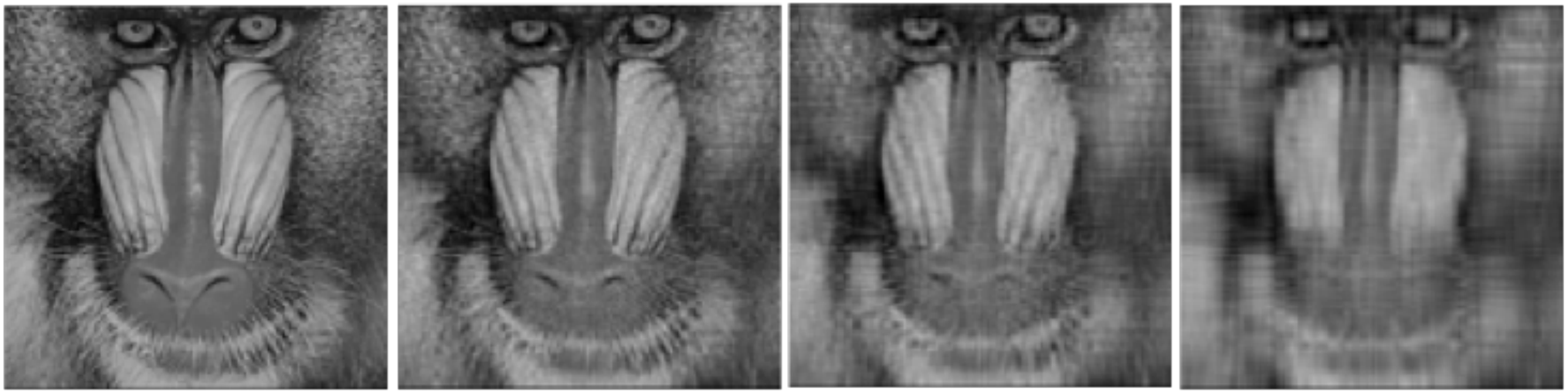
- Expressing the error in approximation as:

$$\varepsilon^2 = \sum_{i=1}^{m} \sum_{j=1}^{n} | f(i,j) - \hat{f}(i,j) |^2$$

- It is possible to show:

$$\varepsilon^2 = \sum_{p=k+1}^{r} \lambda_p$$

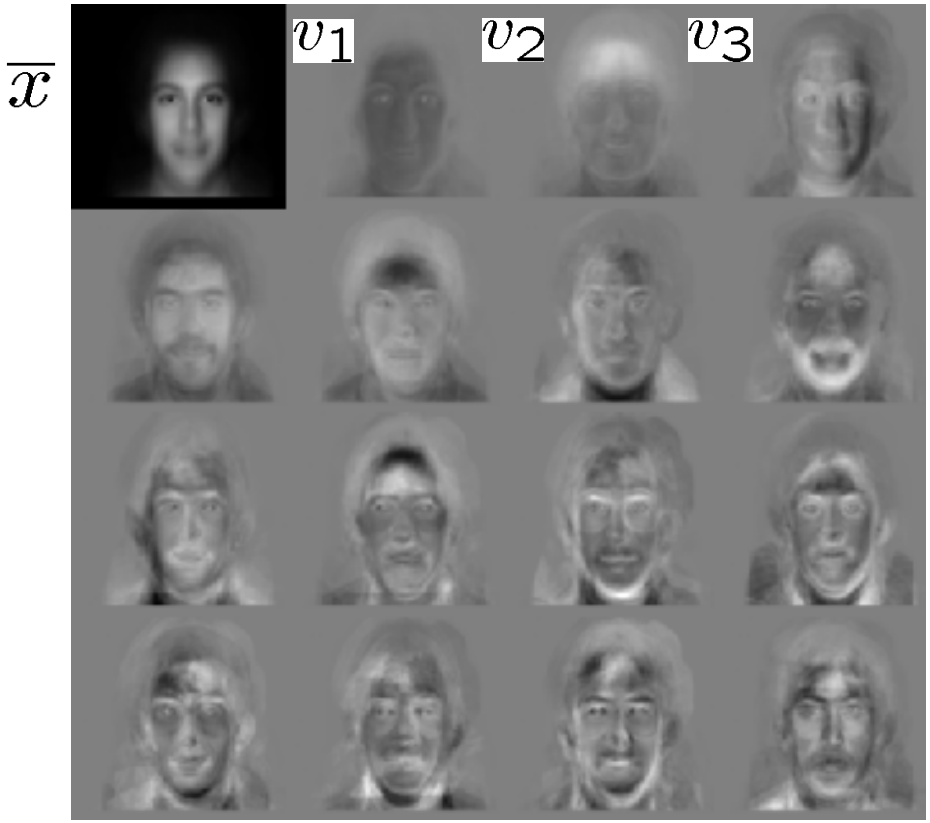
Data Reduction using SVD



Original and three SVD representations using 32, 16, and 8 basis vectors

Eigenfaces

- PCA extracts the eigenvectors of **A**
  - Gives a set of vectors $\mathbf{v_1}$, $\mathbf{v_2}$, $\mathbf{v_3}$, ...
  - Each one of these vectors is a direction in face space
    - what do these look like?

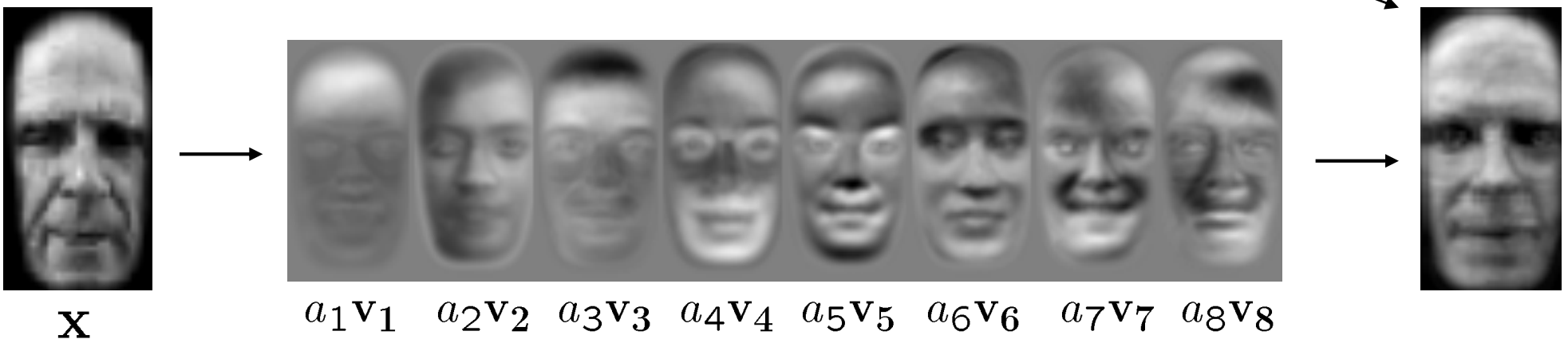


Projecting onto the eigenfaces

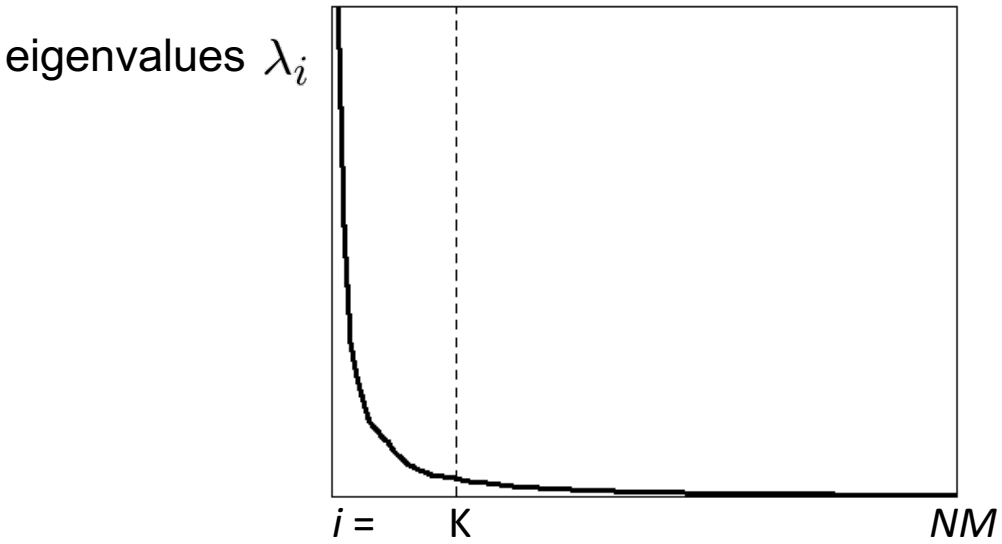
- The eigenfaces $\mathbf{v_1}$, ..., $\mathbf{v_K}$ span the space of faces
  - A face is converted to eigenface coordinates by

$$\mathbf{x} \rightarrow (\underbrace{(\mathbf{x}-\overline{\mathbf{x}})\cdot\mathbf{v_1}}_{a_1}, \underbrace{(\mathbf{x}-\overline{\mathbf{x}})\cdot\mathbf{v_2}}_{a_2}, \ldots, \underbrace{(\mathbf{x}-\overline{\mathbf{x}})\cdot\mathbf{v_K}}_{a_K})$$

$$\mathbf{x} \approx \overline{\mathbf{x}} + a_1\mathbf{v_1} + a_2\mathbf{v_2} + \ldots + a_K\mathbf{v_K}$$



Choosing the dimension K



- How many eigenfaces to use?
- Look at the decay of the eigenvalues
  - the eigenvalue tells you the amount of variance “in the direction” of that eigenface
  - ignore eigenfaces with low variance

PCA Summary

- PCA is a good technique for data reduction
- It may or may not provide good results for classification because its formulation doesn't take into account class separability.
- Fisher's discriminant function, supervised PCA, CCA (Canonical Correlation Analysis) and PLS (Partial Least Squares) are some methods for use with class labels
- PCA is good for linear problems only. Kernel PCA is used for nonlinear problems.
- It is a better practice to normalize data (reduce each variable to zero mean and unit standard deviation) before applying PCA when scales of different features are widely different.

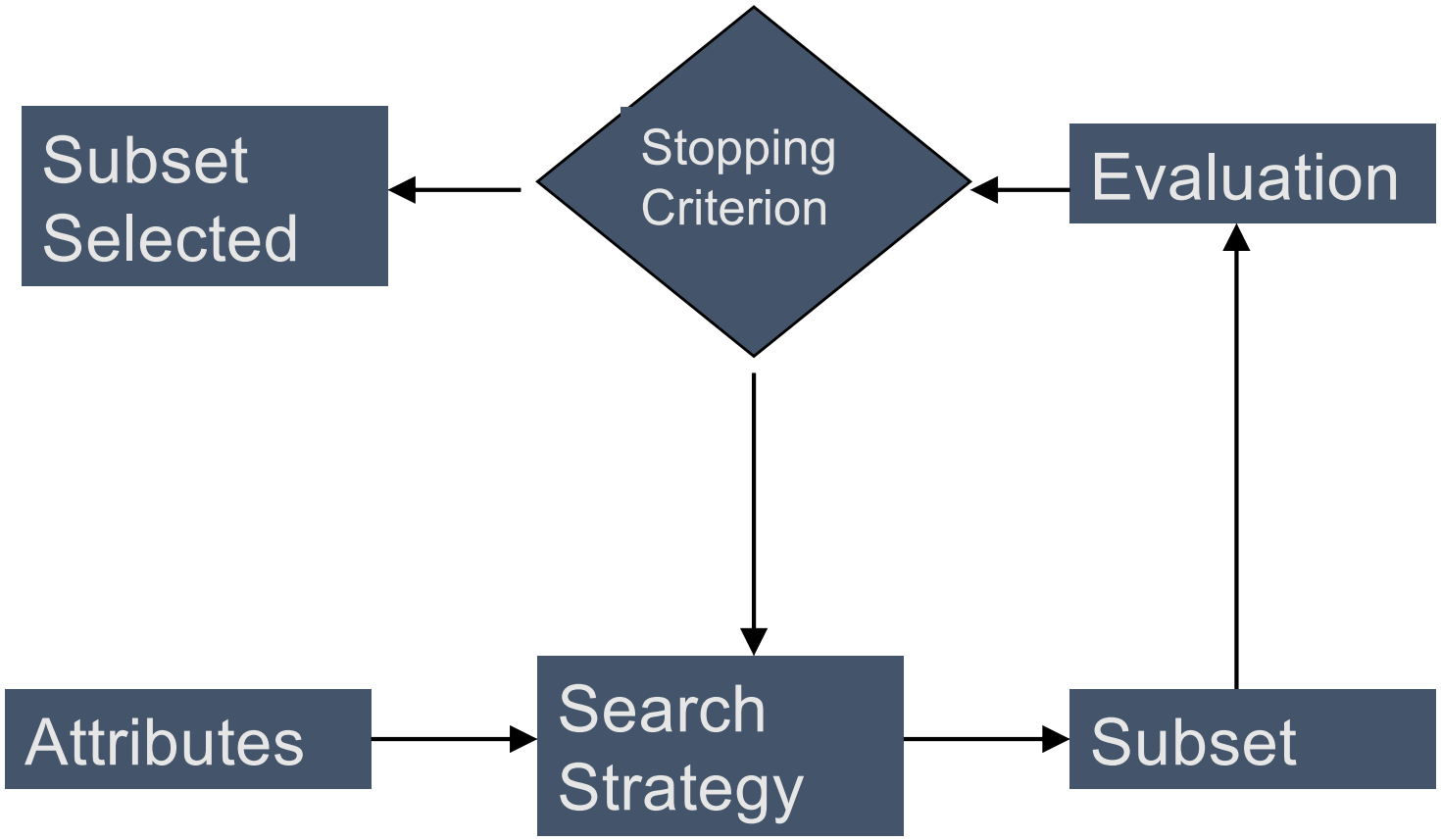
Dimensionality Reduction Via Feature Subset Selection

- The idea here is to select a subset of *k* features out of *n* original features. The number of possible subsets is

  $$n!/k!(n-k)!$$

- For n = 20, k = 5, the number of possible subsets is 15504, an impractical number for exhaustive search

An Approach to Feature Subset Selection Process



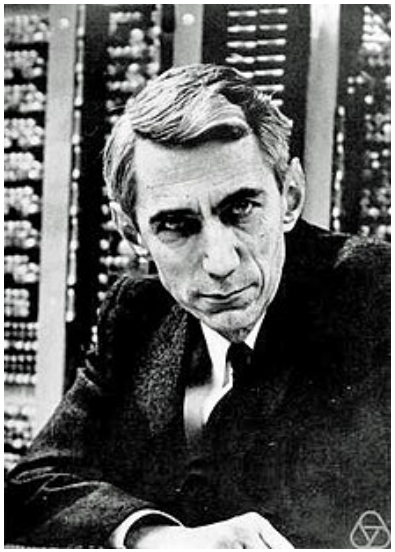
How to Evaluate a Feature Subset

- Use a measure incorporating class discrimination because class separation is our ultimate aim
- Two such measures are:
  - Divergence
  - Scatter matrices based measures

Entropy and Divergence

- Entropy is a measure of uncertainty.
  - A system/event with no uncertainty has zero entropy
  - Entropy of a system/event goes up with increasing uncertainty.
- Consider an event with *n* possible outcomes with p_i being the probability of the *i-th* outcome. The entropy *H* of the event is then defined as:

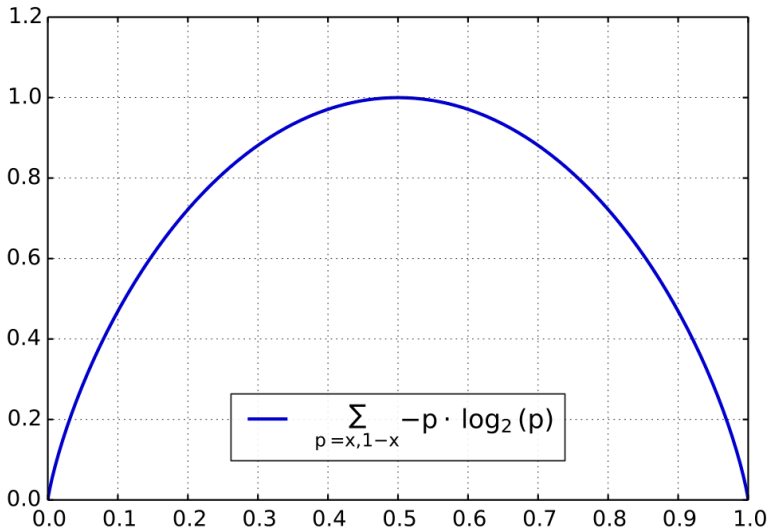
$$H \propto -\sum_{i=1}^{n} p_i \log p_i = \sum_{i=1}^{n} p_i \log \frac{1}{p_i}$$



Claude Shannon

Entropy

- Entropy *H* is 0 if and only if exactly one event has probability 1 and the rest have probability 0. (Uncertainty vanishes only when we are certain about the outcomes.)
- Entropy *H* is maximized when the p_i values are equal.
- The joint entropy of two events is less than or equal the sum of the individual entropies. $H(x,y)=H(x)+H(y)$ only when *x* and *y* are independent events.



Kullback and Leibler (KL) Divergence

- Given two discrete probability distributions, KL divergence measures their similarity. Its defined as

$$D_{\mathrm{KL}}(p\ |q) = \sum_i p_i \log \frac{p_i}{q_i}$$

Also known as *cross entropy*

- D_{KL} is non-negative and zero if and only if $p_i=q_i$ for all i. Note the measure is not symmetric:

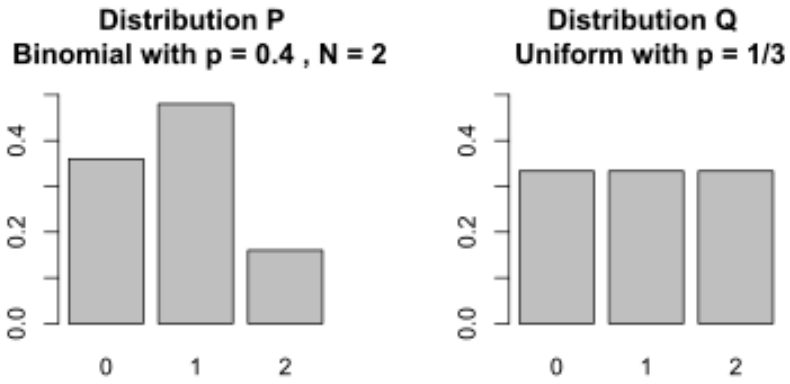
$$D_{\mathrm{KL}}(p|q) \neq D_{\mathrm{KL}}(q|p)$$

- To make the measure symmetric, we define the divergence as

$$D_{\mathrm{KL}}(p,q) = D_{\mathrm{KL}}(p|q) + D_{\mathrm{KL}}(q|p)$$

Example

x	0	1	2
Distribution *P*(x)	0.36	0.48	0.16
Distribution *Q*(x)	0.333	0.333	0.333



$$D_{\mathrm{KL}}(p|q) = 0.36 \ln\left(\frac{0.36}{0.333}\right) + 0.48 \ln\left(\frac{0.48}{0.333}\right) + 0.16 \ln\left(\frac{0.16}{0.333}\right)$$
$$= 0.0852996$$

$$D_{\mathrm{KL}}(q|p) = 0.333 \ln\left(\frac{0.333}{0.36}\right) + 0.333 \ln\left(\frac{0.333}{0.48}\right) + 0.333 \ln\left(\frac{0.333}{0.16}\right)$$
$$= 0.097455$$

KL Divergence Formula for Normal Distributions

- Given two normal distributions, the formula for divergence is:

$$D_{\mathrm{KL}} = \frac{1}{2}\mathrm{trace}[C_1^{-1}C_2 + C_2^{-1}C_1 - 2I] + \frac{1}{2}(M_1 - M_2)^T(C_1^{-1} + C_2^{-1})(M_1 - M_2)$$

- *I* is identity matrix

- M_1 = [3, 3], C1 = 0.2*I*; M_2 = [2.3, 2.3], C2 = 1.9*I*

```
import numpy as np
M1 = np.array([3,3])
C1 = np.array([[0.2, 0],[0, 0.2]])
M2 = np.array([2.3,2.3])
C2 = np.array([[1.9, 0],[0, 1.9]])
I  = np.array([[1,0], [0, 1]])
```

```
C1invC2 = np.linalg.inv(C1)@C2
C2invC1 = np.linalg.inv(C2)@C1
firstTerm = 0.5*np.trace(C1invC2+C2invC1 - 2*I)
```

```
M1minusM2 = M1-M2
C1invplusC2inv = np.linalg.inv(C1)+np.linalg.inv(C2)
secondTerm = 0.5*M1minusM2.T@C1invplusC2inv@M1minusM2
```

```
div = firstTerm+secondTerm
print(div)
```

```
10.313157894736843
```

Scatter Matrices Based Measures

- Class scatter matrix

$$\mathbf{S}_j = \sum_{\mathbf{x}_i \in c_j}^{n_j} (\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^T \text{ for } j = 1, 2$$

where $\mathbf{m}_j = \frac{1}{n_j} \sum_{\mathbf{x}_i \in c_j}^{n_j} \mathbf{x}_i, j = 1, 2$

- Within class scatter matrix $\mathbf{S}_W = \mathbf{S}_1 + \mathbf{S}_2$
- Between class scatter matrix $\mathbf{S}_b = (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T$
- Mixture scatter matrix $\mathbf{S}_m = \mathbf{S}_W + \mathbf{S}_b$

Scatter Matrices Based Measures

$$J_1 = \frac{trace[\mathbf{S}_m]}{trace[\mathbf{S}_w]}$$

$$J_2 = \frac{|\mathbf{S}_m|}{|\mathbf{S}_w|}$$

$$J_3 = trace[\mathbf{S}_w^{-1}\mathbf{S}_m]$$

These measures take on high values when data from two classes are far apart but are clustered together within each class

Feature Subset Search Strategy: Sequential Backward Selection

Let x_1, x_2, x_3, x_4 the available features (m=4). The procedure consists of the following steps:

- Adopt a class separability criterion (could also be the error rate of the respective classifier). Compute its value for ALL features considered jointly $[x_1, x_2, x_3, x_4]^T$.
- Eliminate one feature and for each of the possible resulting combinations, that is $[x_1, x_2, x_3]^T$, $[x_1, x_2, x_4]^T$, $[x_1, x_3, x_4]^T$, $[x_2, x_3, x_4]^T$, compute the class separability criterion value C. Select the best combination, say $[x_1, x_2, x_3]^T$.
- From the above selected feature vector eliminate one feature and for each of the resulting combinations compute and select the best combination.
- Repeat if necessary

The backward selection procedure shows how one can start from *m* features and end up with the "best" *l* ones. The choice is suboptimal. The number of required calculations is:

$$1+\frac{1}{2}\left((m+1)m-\ell(\ell+1)\right)$$

In contrast, a full search requires:

$$\binom{m}{\ell}=\frac{m!}{\ell!(m-\ell)!}$$

operations.

Sequential Forward Selection Strategy

- Here we start by finding the "best" single feature, say x_1
  - For all possible $2\mathrm{D}$ combinations of x_1, i.e., $[x_1, x_2]$, $[x_1, x_3]$, $[x_1, x_4]$ compute the evaluation metric and choose the best, say $[x_1, x_3]$.
  - Repeat for all possible $3\mathrm{D}$ combinations of $[x_1, x_3]$, and choose the best one.
  - The above procedure is repeated till the "best" vector with *l* features has been formed
  - This is also a suboptimal technique, requiring

$$\ell m - \frac{\ell(\ell-1)}{2}$$ operations.

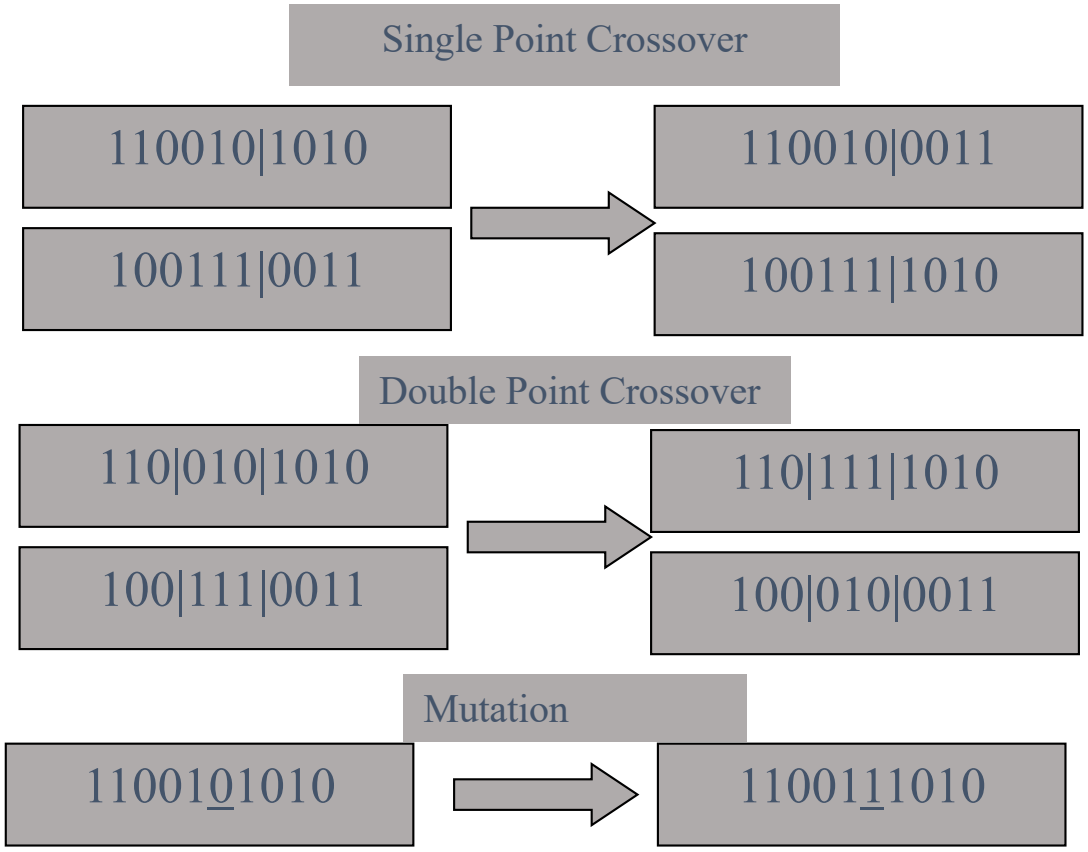
Genetic Algorithms for Feature Reduction

- Derivative-free search methods
- Draw upon the ideas of natural selection and evolution
- Easily realizable on parallel processing machines
- Applicable to discrete and continuous optimization problems
- Less prone to locally minimum solutions

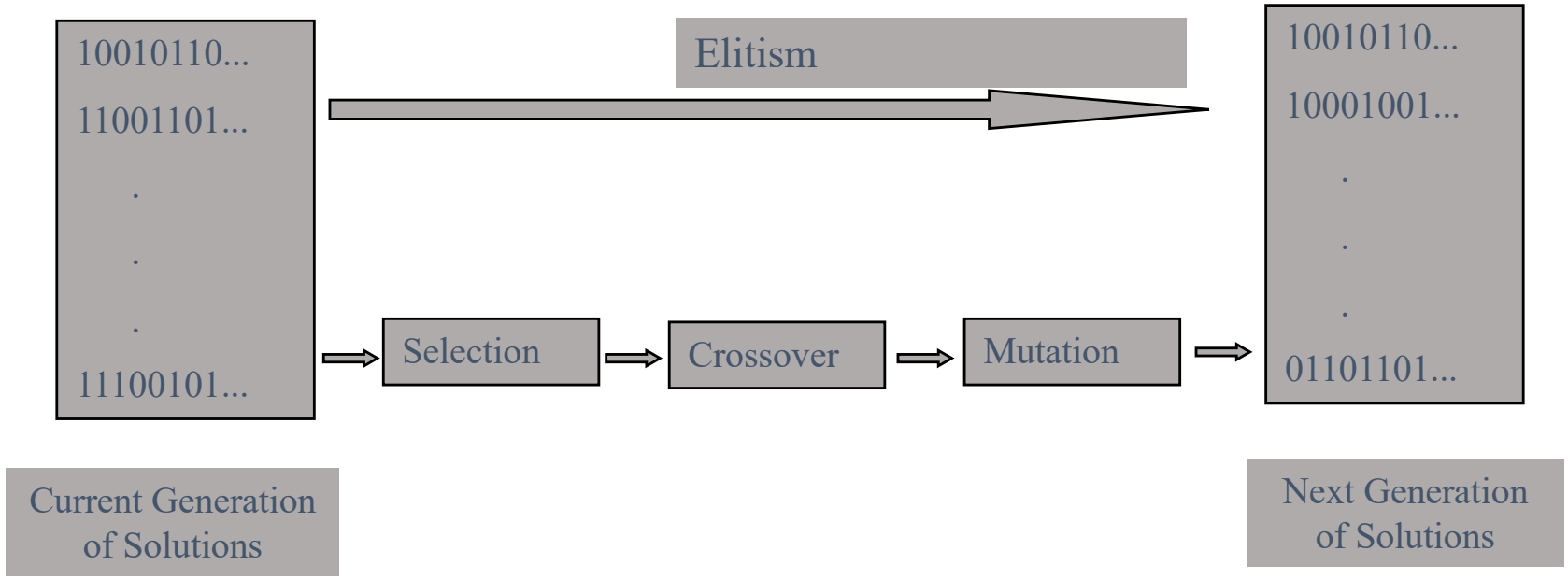
Problem Solving using Genetic Algorithms

- Form a bit string representation for solution space (*chromosome*)
- Define a fitness function
- Form an initial *gene pool*, i.e. initial population of solutions
- Evolve using *genetic operators*

Examples of Genetic Operators



Evolution in Genetic Algorithms



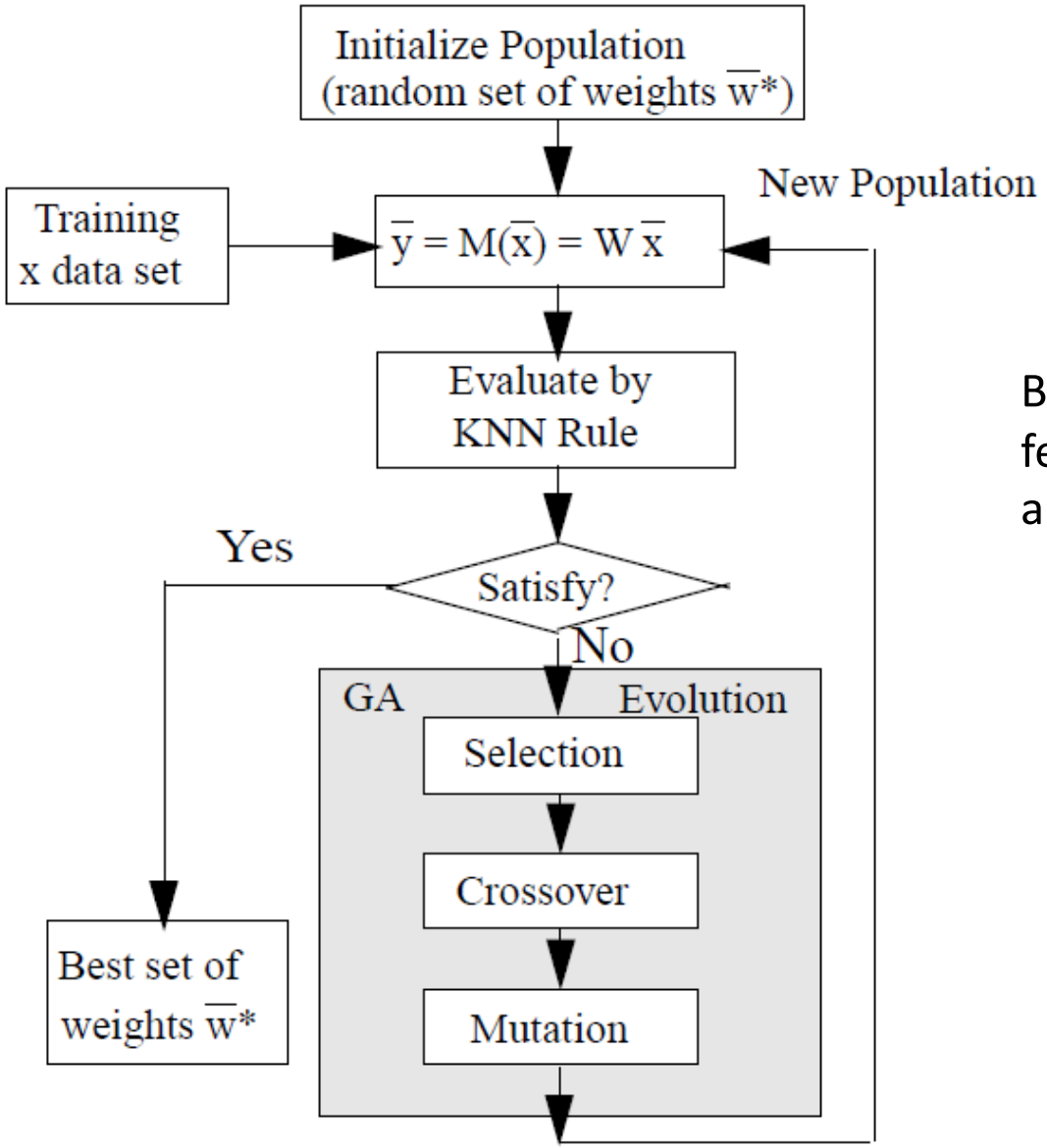
Feature Selection using GA

- Feature set representation?
- Fitness function?

Table 2: Comparison of neural network pattern classifiers constructed using the entire set of attributes against those constructed using the best (in accuracy) GA-selected subset.

	All Attributes			GA-selected Subset		
Dataset	*Dimension*	*Accuracy*	*Hidden*	*Dimension*	*Accuracy*	*Hidden*
3P	13	79.0±12.2	5.0 ± 2.0	6.6 ± 1.6	100 ± 0.0	9.2 ± 4.9
Bridges	11	63.0 ± 7.8	5.2 ± 3.3	5.6 ± 1.5	81.6 ± 7.6	17.6 ± 12.4
Cancer	9	97.8 ± 1.2	2.9 ± 1.2	5.4 ± 1.4	99.3 ± 0.9	5.7 ± 2.9
CRX	15	87.7 ± 3.3	7.7 ± 6.9	8.0 ± 2.1	91.5 ± 2.8	12.5 ± 7.6
Glass	9	70.5 ± 8.5	9.8 ± 6.9	5.5 ±1.4	80.8 ±5.0	14.5 ± 6.6
Heart	13	86.7 ± 7.6	5.7 ± 4.4	7.2 ±1.6	93.9 ±3.8	7.5 ± 3.9
HeartCle	13	85.3 ± 2.7	3.4 ± 1.1	7.3 ±1.7	92.9 ±3.6	7.6 ± 4.2
HeartHun	13	85.9 ± 6.3	5.0 ± 2.9	7.0 ±1.2	93.0 ±4.0	7.1 ± 3.7
HeartSwi	13	94.2 ± 3.8	2.2 ± 0.6	6.6 ±1.7	98.3 ±3.3	3.7 ± 1.5
HeartVa	13	80.0 ± 7.4	5.1 ± 2.6	7.1 ±1.7	91.0 ±5.7	8.5 ± 3.0
Hepatitis	19	84.7 ± 9.5	6.2 ± 4.0	9.2 ±2.3	97.1 ±4.3	8.1 ± 2.8
Horse	22	86.0 ± 3.6	5.3 ± 4.5	11.1 ±2.3	92.6 ±3.4	9.5 ± 4.1
Ionosphere	34	94.3 ± 5.0	5.5 ± 1.6	17.3 ±3.5	98.6 ±2.4	7.5 ± 2.4
Liver	6	72.9 ± 5.1	21.5 ± 27.3	4.1 ±0.7	77.8 ±4.0	25.9 ± 24.3
Pima	8	76.3 ± 5.1	8.1 ± 4.9	3.8 ±1.5	79.5 ±3.1	20.8 ± 21.2
Promoters	57	88.0 ± 7.5	2.2 ± 0.4	28.8 ±3.3	100 ±0.0	2.7 ± 1.0
Sonar	60	83.0 ± 7.8	6.4 ± 2.7	30.7 ±3.7	97.2 ±2.9	7.2 ± 3.0
Votes	16	96.1 ± 1.5	3.2 ± 1.5	8.9 ±1.8	98.8 ±1.2	4.0 ± 1.8
Vowel	10	69.8 ± 6.4	38.0 ± 8.3	6.5 ±1.2	78.4 ±3.8	41.5 ± 7.7
Wine	13	97.1 ± 4.0	5.5 ± 1.7	6.7 ±1.6	99.4 ±2.1	5.9 ± 2.1
Zoo	16	96.0 ± 4.9	6.1 ± 1.1	9.3 ±1.6	100 ±0.0	6.2 ± 1.1

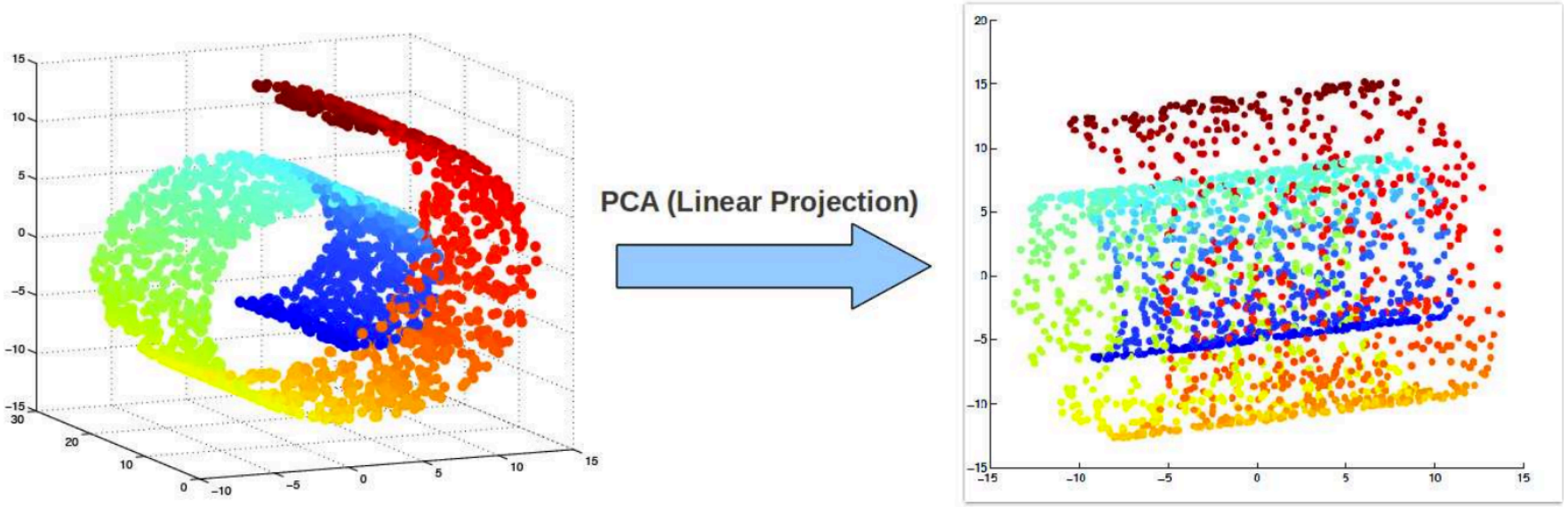
Feature Transformation using GA

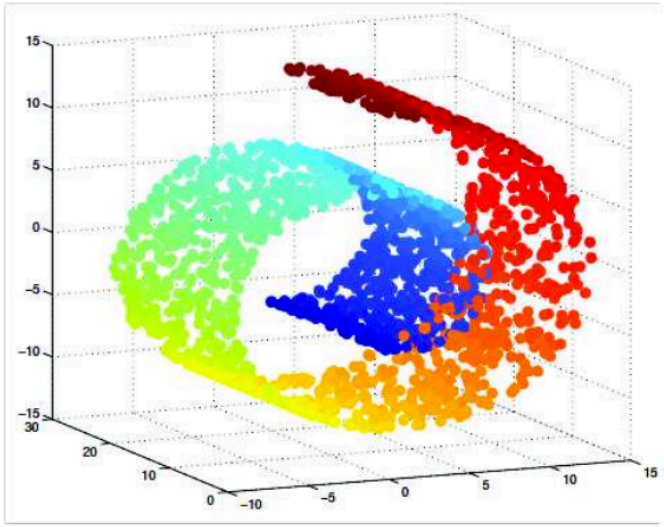


Both linear and nonlinear feature transformations are possible

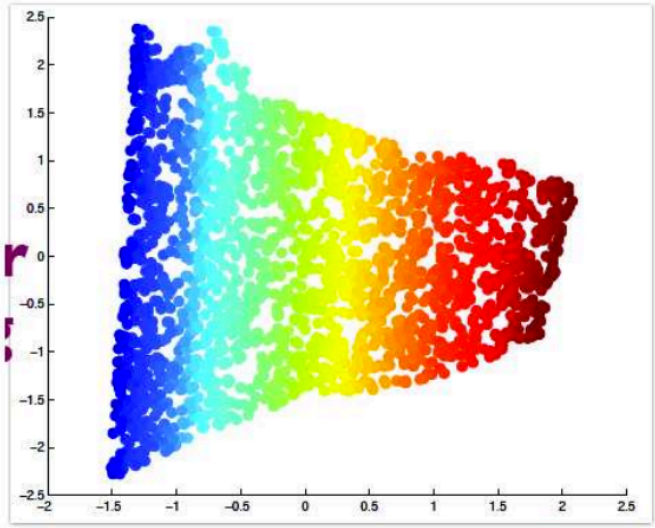
Dimensionality Reduction using Local Embedding

In many cases, data might be lying on a low-dimensional non-linear surface (manifold) in the feature space. In such cases, linear projection via PCA will not produce good results





Swiss-roll data



Unrolled data

A nonlinear projection does a better job of uncovering intrinsic dimensionality in such cases. How to get such a projection?

It turns out that one can get nonlinear projections by considering and preserving neighborhood information. One such recent method is *t-SNE* method which is very popular at present.

t-SNE Method

- It is an extension of SNE (Stochastic Neighborhood Embedding) method.
- The lowercase “t” stands for Student’s-t distribution and the method is called t-Distributed Stochastic Neighborhood Embedding (t-SNE) method.

t-SNE Method

- We have *N* data points in a high-dimensional space $\mathbf{x}_1, \ldots, \mathbf{x}_N$
- Define similarity (p_{ij}) between $\mathbf{x}_i$ and $\mathbf{x}_j$ in the following manner:

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2 / 2\sigma_i^2)},$$

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$$

Performing computations as per above formulas results in a similarity matrix

This expression defines the probability that i-th point would pick the j-th point as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian centered at i-th point.

This is because we want distance from A to B be same as from B to A

σ_i is called *perplexity* parameter. It kind of controls the effective number of neighbors for a given point. This variance is different for every point; it is chosen such that points in dense areas are given a smaller variance than points in sparse areas.

t-SNE Method (Continued)

Let the lower dimensional mapping be represented as $\mathbf{y}_1, \ldots, \mathbf{y}_N$

Lets apply the same idea of measuring similarity in the lower dimensional space.

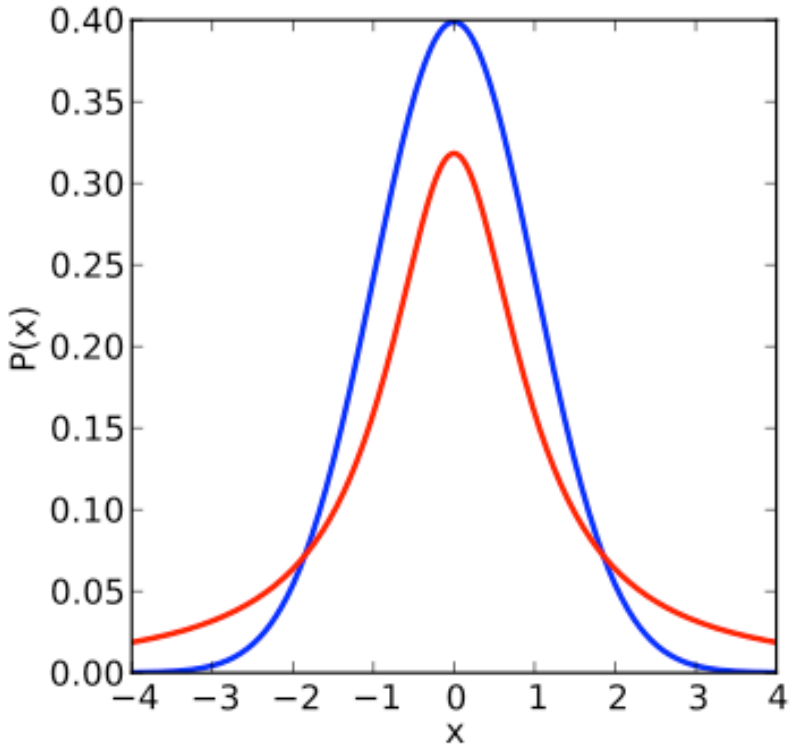
$$q_{ij} = \frac{f(|y_i - y_j|)}{\sum_{k \neq i} f(|y_i - y_k|)} \quad \text{with} \quad f(z) = \frac{1}{1 + z^2}$$

In this case, We use the Students'-t distribution with a single degree of freedom to compute similarities.

In the original SNE method, Gaussian distribution was used. Later on, Students'-t distribution was found to be more appropriate.

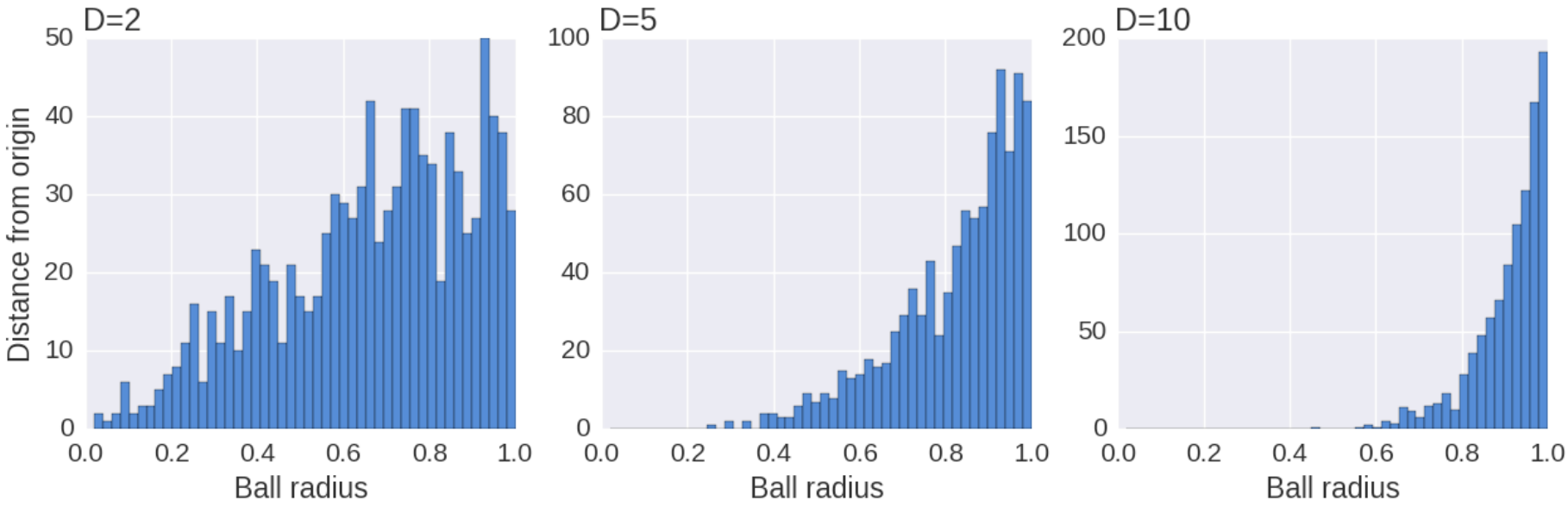
What is Students'-t Distribution?

- The *t*-distribution is symmetric and bell-shaped, like the normal distribution, but has heavier tails, meaning that it is more prone to producing values that fall far from its mean.



Why Use Students'-t Distribution?

- Let's generate uniformly distributed random points within a sphere of increasing dimensions but same radius.
- Let's calculate the histogram of distances of the generated points from origin. What we note from the figure below is that distributions of distances are widely different as we go from low to high dimensions.
- Based on this difference in low and high-dimensional spaces, the Students'-t provides a better fit in the mapped space.



How are Mapped Points Found?

- The points in the original high-dimensional space are fixed. The mapping in the lower dimensional space is found by minimizing the following criteria

$$KL(P||Q) = \sum_{i,j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

Known as Kullback-Leiber divergence

To minimize this score, we perform a gradient descent. The gradient can be computed analytically:

$$\frac{\partial KL(P||Q)}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij}) g\left(|x_i - x_j|\right) u_{ij} \quad \text{where } g(z) = \frac{z}{1+z^2}.$$

Here, u_{ij} is a unit vector going from y_j to y_i.

A selection from the 64-dimensional digits dataset

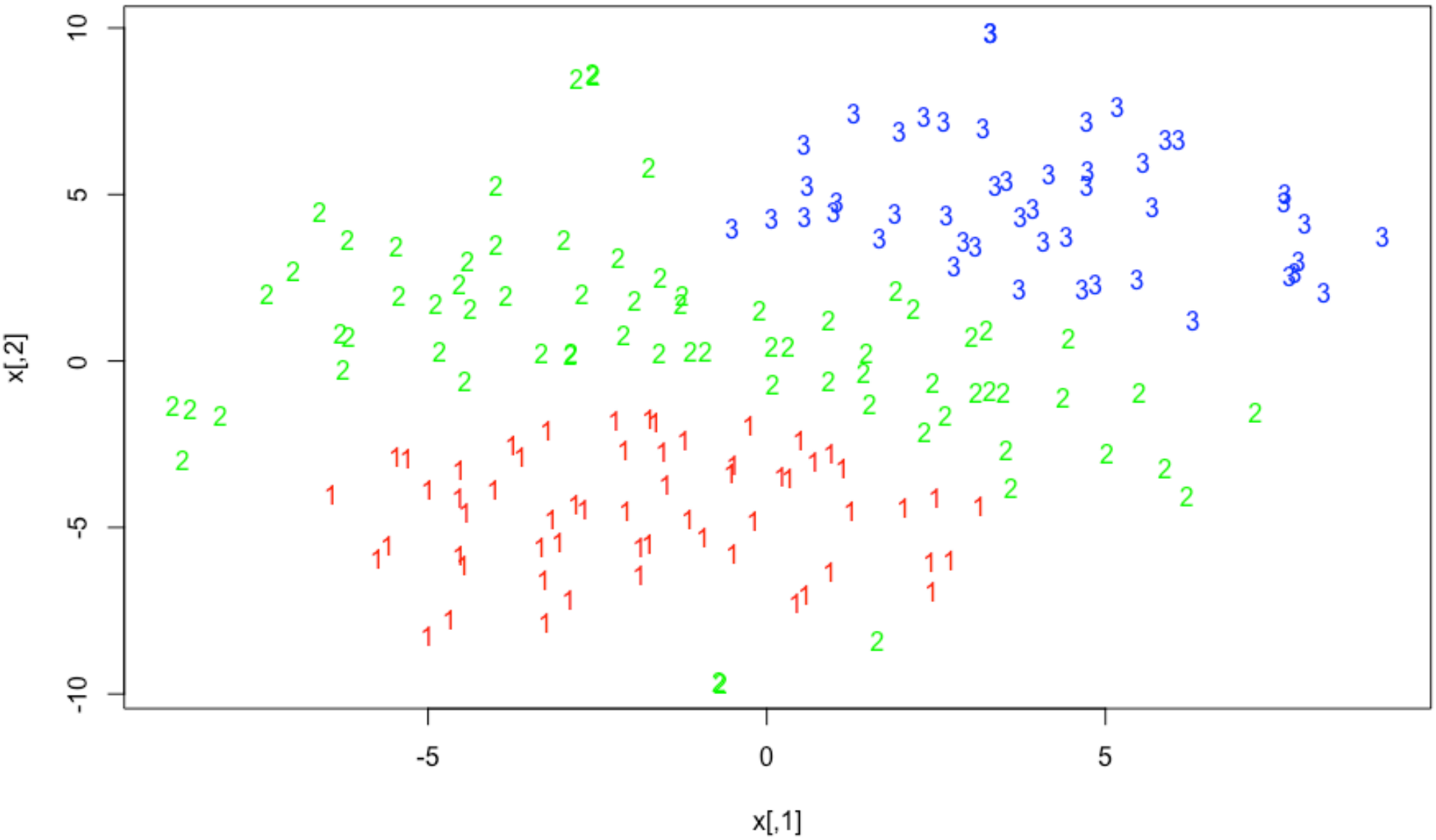
1797 point (digits) in a 8x8 (64-dimensional space)

t-SNE mapping of 1796 points from a 64-dimensional to a 2-dimensional space.

Using t-SNE

- Remember, t-SNE is a gradient search algorithm. You can get slightly different visualizations based on initial configuration of points in the lower-dimensional space.
- Often, mapping in 2-D via PCA is used as an initial configuration.
- The coloration of points in the demo for digits is only for illustration. The algorithm doesn't take into consideration the class labels, if any, into consideration
- The perplexity parameter is automatically determined.

Wine Data Mapping using t-SNE



PCA Mapping of Wine data using the first two eigenvectors

Summary

- Many methods for feature selection and dimensionality reduction
- PCA/Eigen vectors/K-L transform is a popular approach to obtain low dimensional space via linear projection
- t-SNE approach is currently in favor for linear and nonlinear configurations.
- Other methods/approaches not discussed:
  - Isomap
  - Kernel PCA
  - MDS (to be discussed later)
  - LDA/MLDA (to be discussed later)