Machine Learning

By Ghazal Lalooha

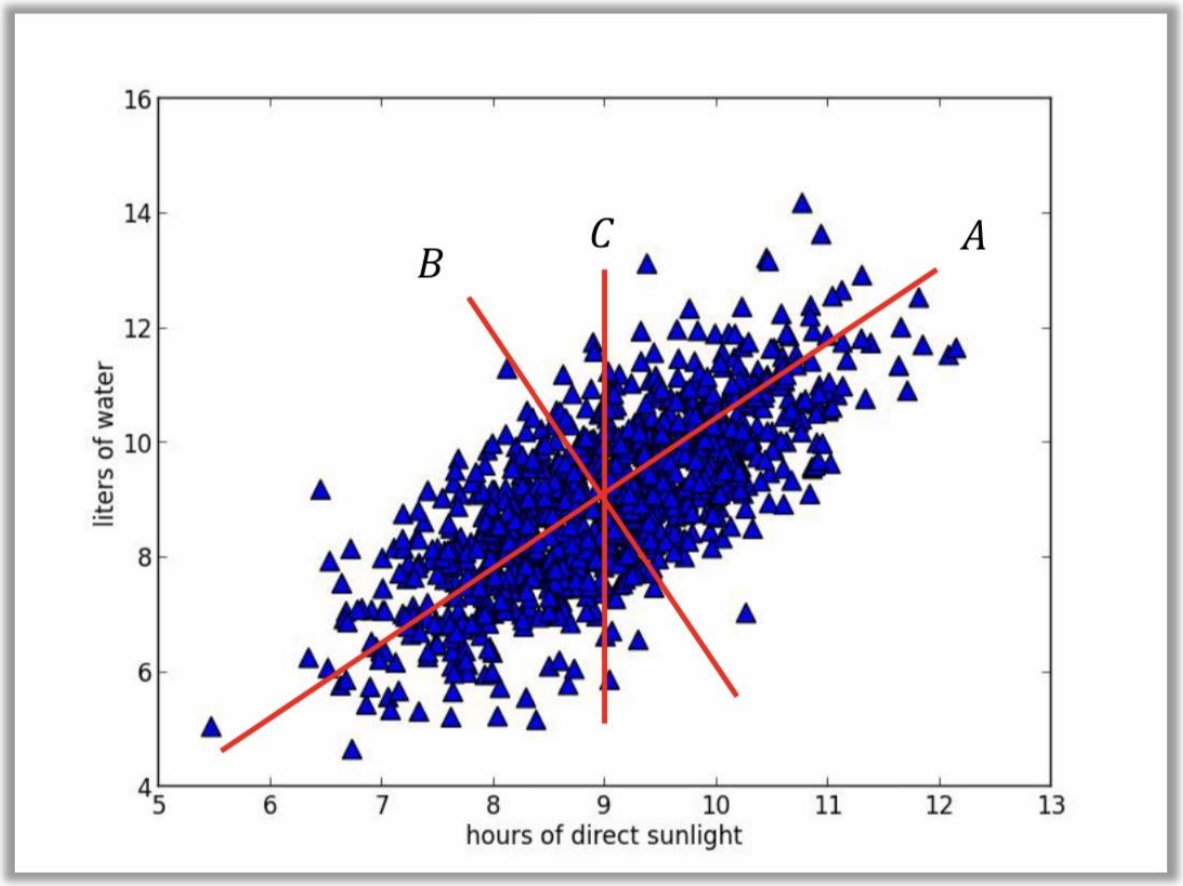
Dimension reduction

Dimension reduction

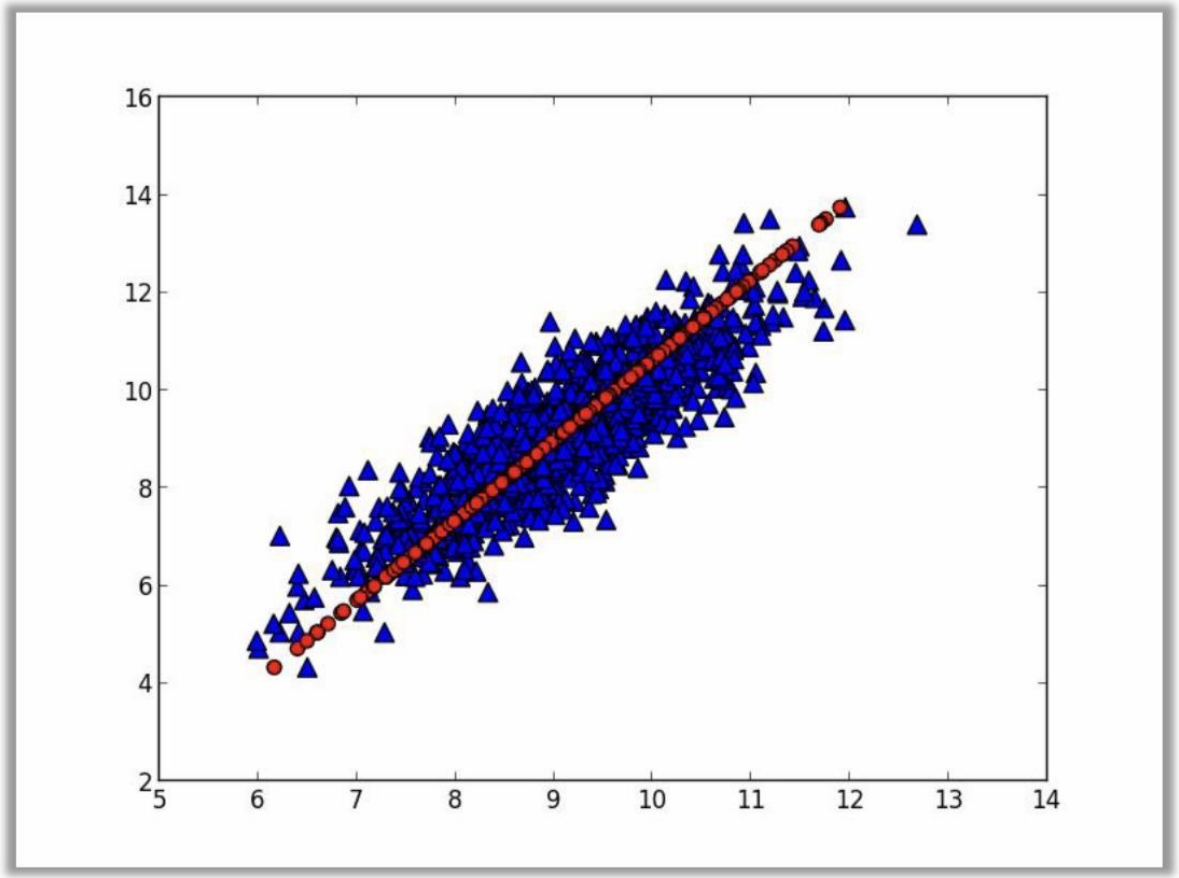
- Motivation:
  - Display data
  - Data compression
    - Reduce memory consumption
    - Easier to use datasets
    - Reducing the computational costs of many algorithms
    - Removing noise and increasing the accuracy of the learning algorithm
    - Make it easier to understand the results
- methods:
  - Principal component analysis
  - Analysis of factors
  - Independent component analysis

Principal component analysis(PCA)

Principal component analysis



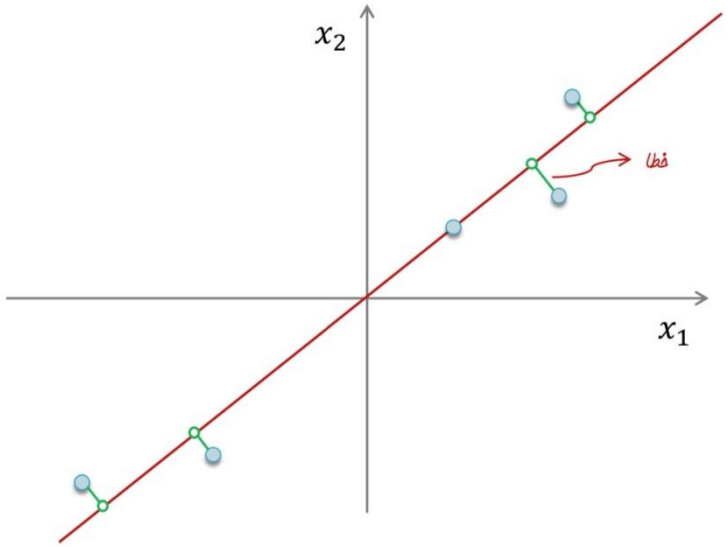
Principal component analysis



Principal component analysis: problem statement

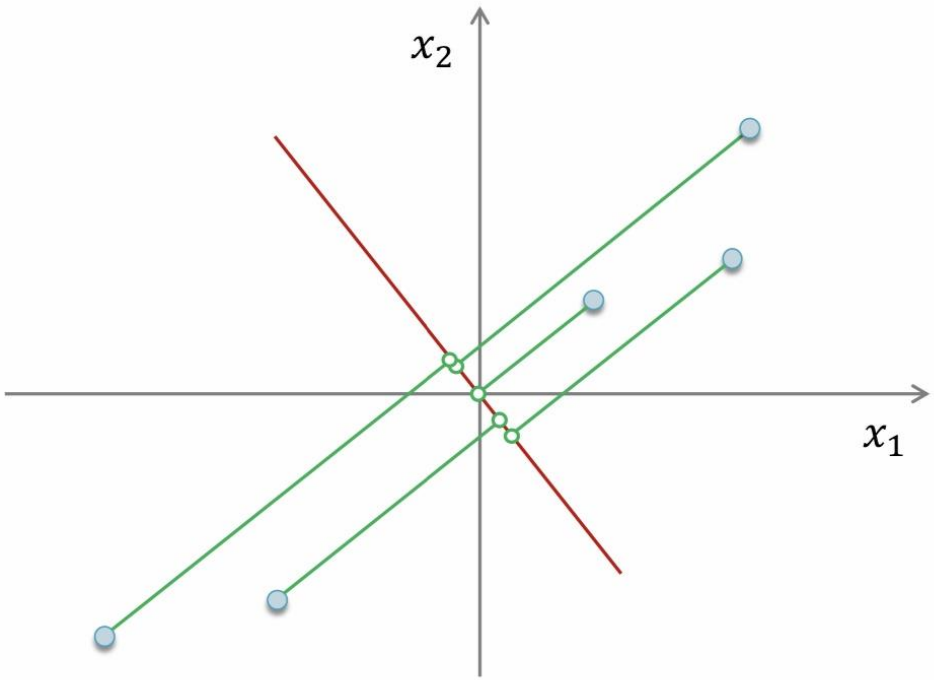
problem statement

- Objective: Minimization of the sum of squares of radiation error

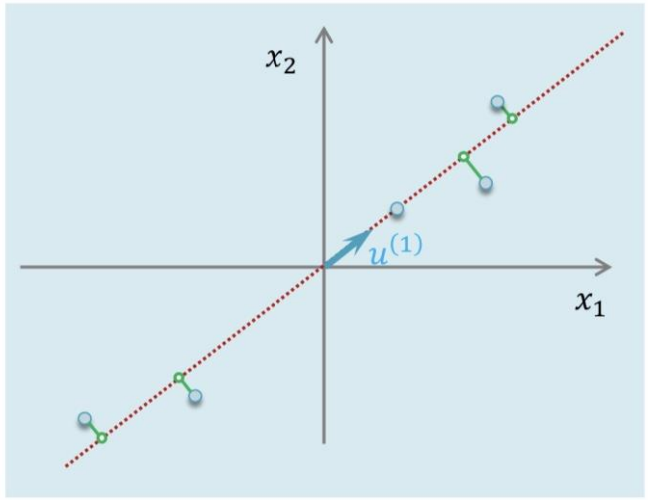


problem statement

- Objective: Minimization of the sum of squares of radiation error

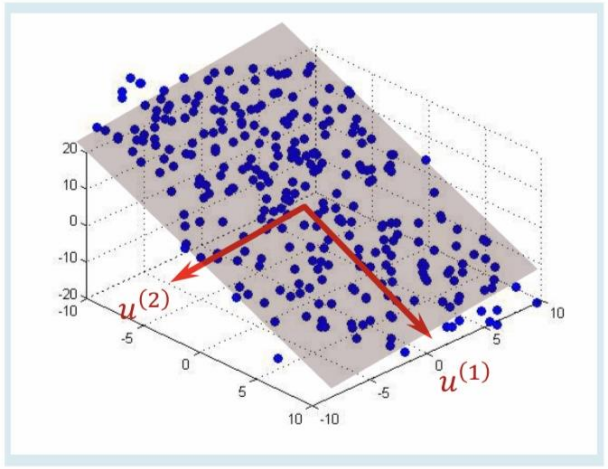


problem statement



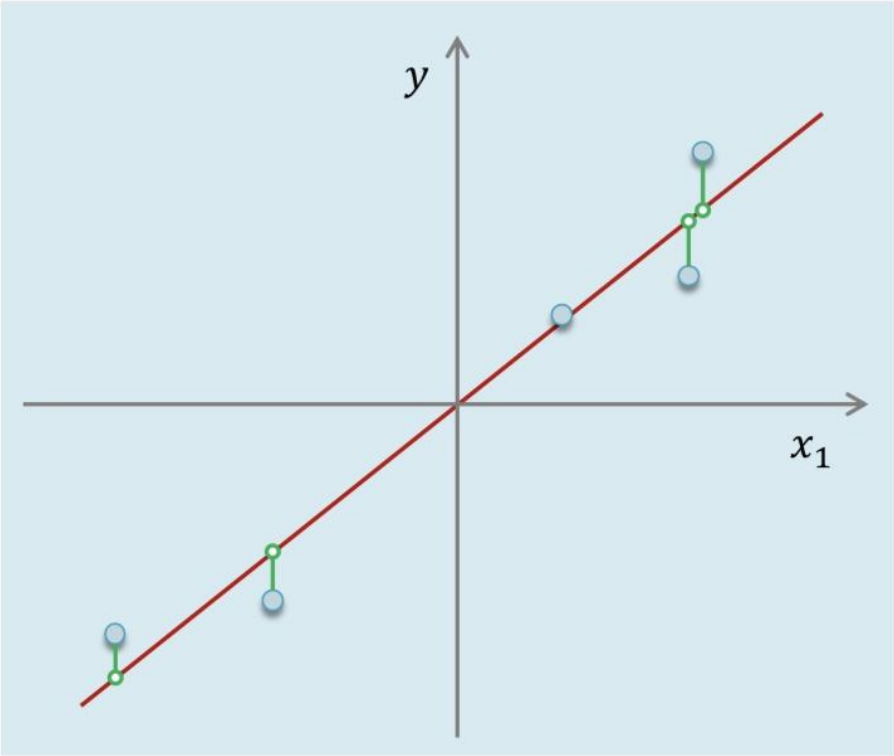
- Dimension reduction from 2 to 1:
  - Finding a direction such as u $^{(1)}$ (where u is a real vector) so that by plotting the points in that direction, the sum of squared errors is minimized.

problem statement

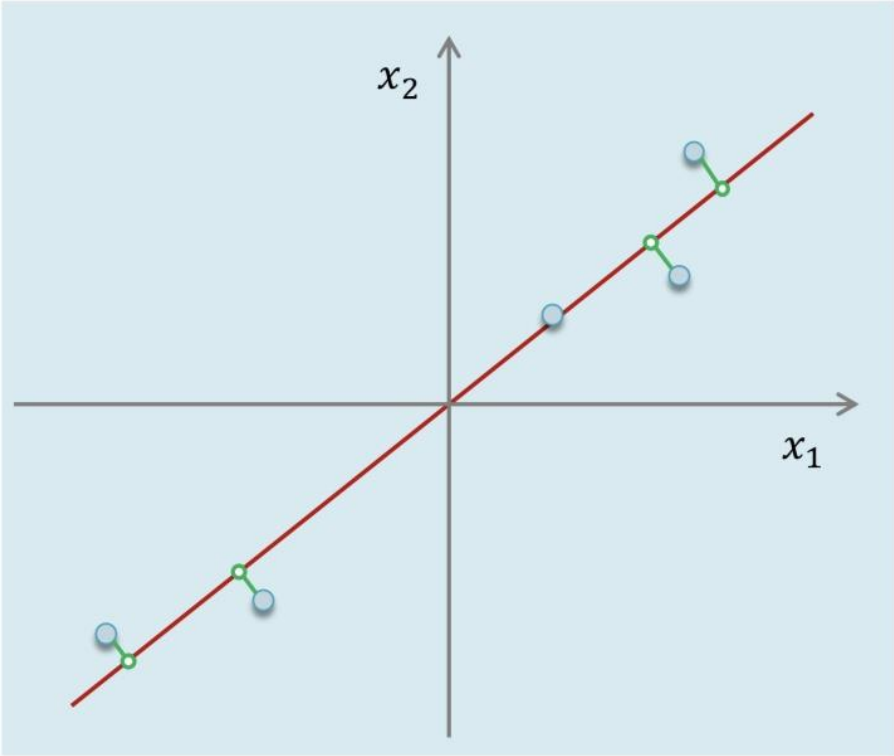


- Dimension reduction from d to k:
  - Finding k orthogonal vectors such as u $^{(1)}$, u $^{(2)}$, ..., u $^{(k)}$ (where u is a real vector) so that the sum of squared errors is minimized by depicting the points in that direction.

Is PCA the same as regression?



Linear regression



PCA

PCA algorithm

PCA algorithm: Pre-Processing

- Training set: $\{x^{(1)}, x^{(2)}, x^{(3)}, \cdots, x^{(m)}\}, \qquad x^{(i)} \in \mathbb{R}^n$

- Pre-Processing:
  - Mean removing:

$$\mu_j = \frac{1}{m}\sum_{i=0}^{m} x_j^{(i)}, \qquad x_j^{(i)} = x_j^{(i)} - \mu_j$$

  - Scaling (if necessary):
    - If different features have different ranges (such as the features of house size and number of rooms), scale the features so that the ranges of the different features are approximately equal.

$$x_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{s_j}$$

s_j ← Standard deviation in attribute j

PCA algorithm

- Reducing data dimensions from d to k:
  - Calculation of the covariance matrix:

$$\Sigma = \frac{1}{m} X^T X = \frac{1}{m} \sum_{i=1}^{n} x^{(i)} \left(x^{(i)}\right)^T$$

  - Calculation of eigenvectors of the covariance matrix:

$$[U, S, V] = svd(\Sigma)$$

  - Choosing the first k vector from the matrix U:

$$U = \begin{bmatrix} | & | & \cdots & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(n)} \\ | & | & \cdots & | \end{bmatrix}_{n \times n}$$

$$U_{reduced} = \begin{bmatrix} | & | & \cdots & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(k)} \\ | & | & \cdots & | \end{bmatrix}_{n \times k}$$

PCA algorithm

- Calculation of new data with k dimensions:

$$z_{k\times 1}^{(i)} = \begin{bmatrix} | & | & \cdots & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(k)} \\ | & | & \cdots & | \end{bmatrix}^T \times x_{n\times 1}^{(i)}$$

$$= \begin{bmatrix} - & u^{(1)} & - \\ - & u^{(2)} & - \\ \vdots & \vdots & \vdots \\ - & u^{(k)} & - \end{bmatrix}_{k\times n} \times x_{n\times 1}^{(i)}$$

Implementation in Oktav

- PCA algorithm:
  - After removing the mean and if needed scaling

```
function Z = PCA(X)

        m = size(X, 1);

        Sigma = (1/m) * X' * X;

        [U, S, V] = svd(Sigma);

        U_reduced = U(:, 1:k);

        Z = X * U_reduced;

end;
```

Calculate the covariance matrix

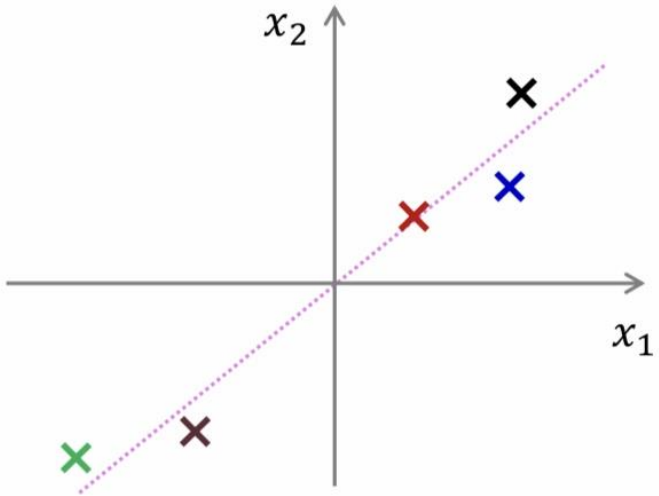
Calculate the decomposition of single values

Select the first k component

Compute new data with k dimensions.

Example: Dimension reduction

$$Z = X \times U_{reduced}$$

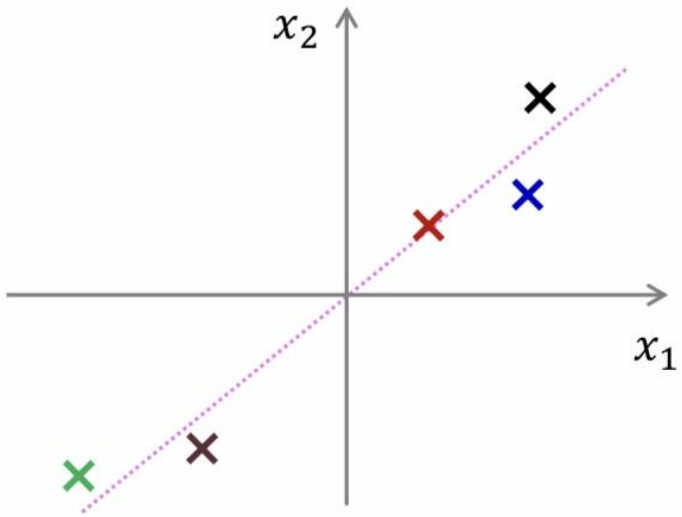


Initial data (2-dimension)

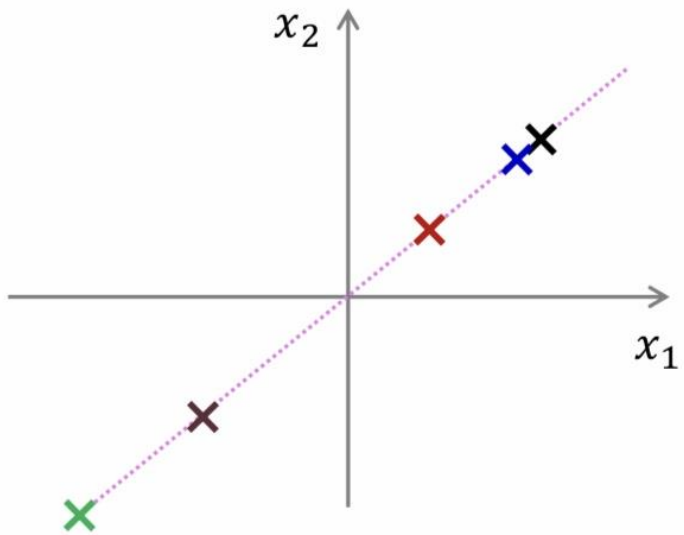
new data (1-dimension)

Example: Dimension reduction

$$X_{recovered} = Z * U^T_{reduced} + means$$



Initial data (2-dimension)



Reconstruction data

PCA algorithm : mean removing

```
>> X = [1 1 1 0 0; 2 2 2 0 0; 1 1 1 0 0; 5 5 5 0 0;
       1 1 0 2 2; 0 0 0 3 3; 0 0 0 1 1];
>> meanVals = mean(X);
>> X_norm = zeros(size(X));
>> for i=1:size(X, 1)
>       X_norm(i, :) = X(i, :) - meanVals;
>> X_norm
X_norm =
  -0.42857  -0.42857  -0.28571  -0.85714  -0.85714
   0.57143   0.57143   0.71429  -0.85714  -0.85714
  -0.42857  -0.42857  -0.28571  -0.85714  -0.85714
   3.57143   3.57143   3.71429  -0.85714  -0.85714
  -0.42857  -0.42857  -1.28571   1.14286   1.14286
  -1.42857  -1.42857  -1.28571   2.14286   2.14286
  -1.42857  -1.42857  -1.28571   0.14286   0.14286
```

Initial data

1	1	1	0	0
2	2	2	0	0
1	1	1	0	0
5	5	5	0	0
1	1	0	2	2
0	0	0	3	3
0	0	0	1	1

```
X_norm = bsxfun(@minus, X, mean(X))
```

PCA Algorithm: Calculation of eigenvectors

```
>> m = size(X, 1);
>> Sigma = (1/m) * X_nom' * X_norm;

>> [U, S, V] = svd(Sigma);
>> S
S =
Diagonal Matrix
  8.7173e+000            0            0            0            0
            0  1.5832e+000            0            0            0
            0            0  6.6876e-002            0            0
            0            0            0  5.6022e-016            0
            0            0            0            0  1.4144e-047
```

PCA Algorithm: Dimension reduction

```
>> U_red = U(:, 1:3);

>> Z = X_norm * U_red;
Z =
   0.1667425   1.3749474  -0.0091539
  -1.4442884   0.7390287  -0.0228180
   0.1667425   1.3749474  -0.0091539
  -6.2773812  -1.1687275  -0.0638103
   1.7595299  -1.1001502   0.5712943
   3.3326043  -1.9204675  -0.3520239
   2.2960504   0.7004216  -0.1143345
```

PCA Algorithm: Reconstruction of primary data

```
>> X_recovered = Z * U_red';
>> for i = 1:m
>     X_recovered(i, :) = X_recovered(i, :) + meanVals;
>> X_recovered
X_recovered =
   1.00000   1.00000   1.00000   0.00000   0.00000
   2.00000   2.00000   2.00000   0.00000   0.00000
   1.00000   1.00000   1.00000   0.00000   0.00000
   5.00000   5.00000   5.00000  -0.00000  -0.00000
   1.00000   1.00000   0.00000   2.00000   2.00000
   0.00000  -0.00000   0.00000   3.00000   3.00000
  -0.00000  -0.00000  -0.00000   1.00000   1.00000
```

Initial data

1	1	1	0	0
2	2	2	0	0
1	1	1	0	0
5	5	5	0	0
1	1	0	2	2
0	0	0	3	3
0	0	0	1	1

Selection of the number of main components

Selection of the number of main components

- Average sum of squares of radiation error:

Keeping 99% of the variance

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}-x_{approx}^{(i)}\right\|^2}{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}\right\|^2}\leq 0.01$$

Keeping 95% of the variance

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}-x_{approx}^{(i)}\right\|^2}{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}\right\|^2}\leq 0.05$$

Keeping 90% of the variance

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}-x_{approx}^{(i)}\right\|^2}{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}\right\|^2}\leq 0.10$$

Selection of the number of main components

An inefficient algorithm

$k = 0$

repeat

{

$k = k + 1$

try $PCA(X)$ with k components

compute $U_{reduced}, z^{(1)}, z^{(2)}, \dots, z^{(m)}, x_{approx}^{(1)}, x_{approx}^{(2)}, \dots, x_{approx}^{(m)}$

} **until** $\frac{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)} - x_{approx}^{(i)}\right\|^2}{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}\right\|^2} \leq 0.01$

Selection of the number of main components

remove mean from X

$$sigma = \frac{1}{m}(X' * X);$$

$$[U, S, V] = \text{svd}(sigma);$$

pick smallest value of k for which:

$$\frac{\sum_{i=1}^{k} s_{ii}}{\sum_{i=1}^{n} s_{ii}} \geq 0.99$$

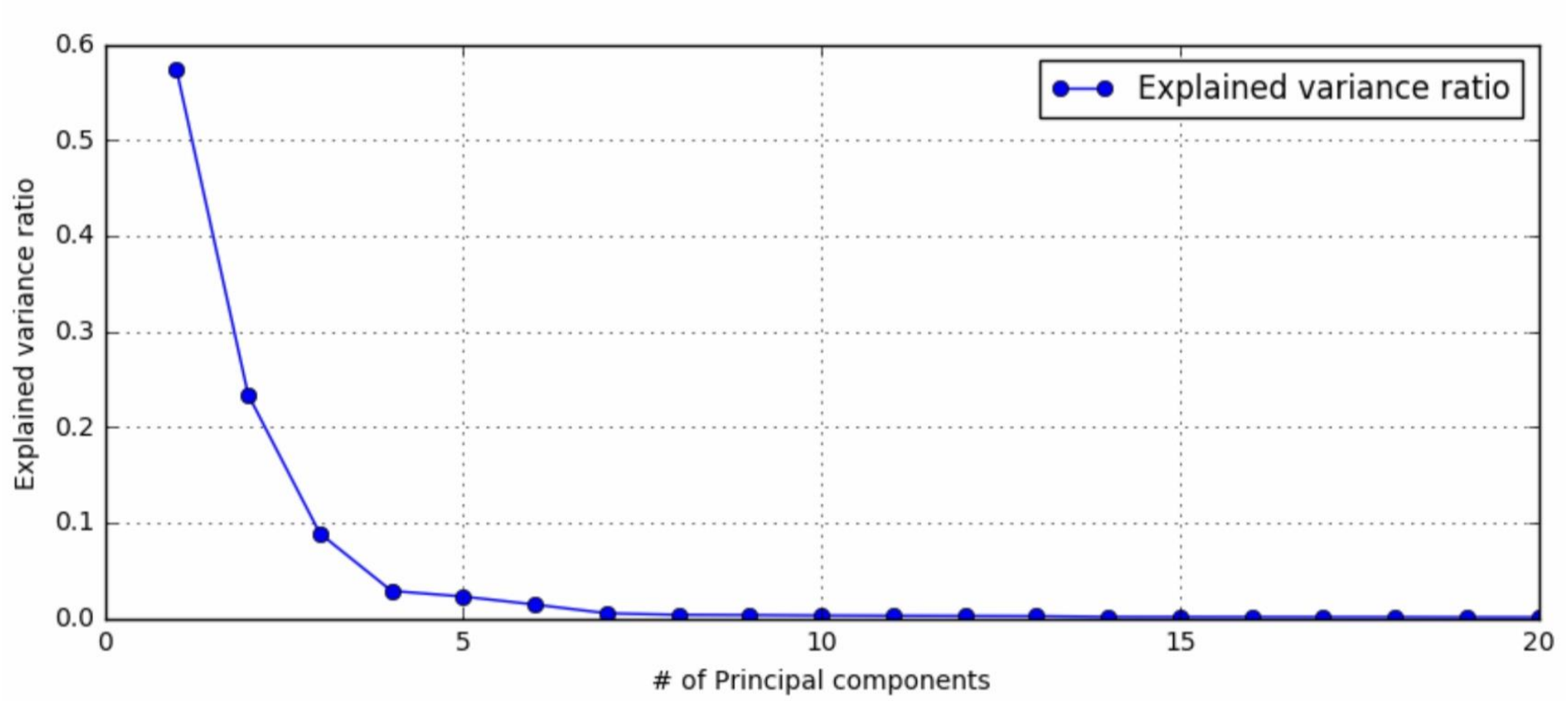
$$S = \begin{bmatrix} s_{11} & 0 & \cdots & 0 \\ 0 & s_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & s_{nn} \end{bmatrix}$$

Selection of the number of main components

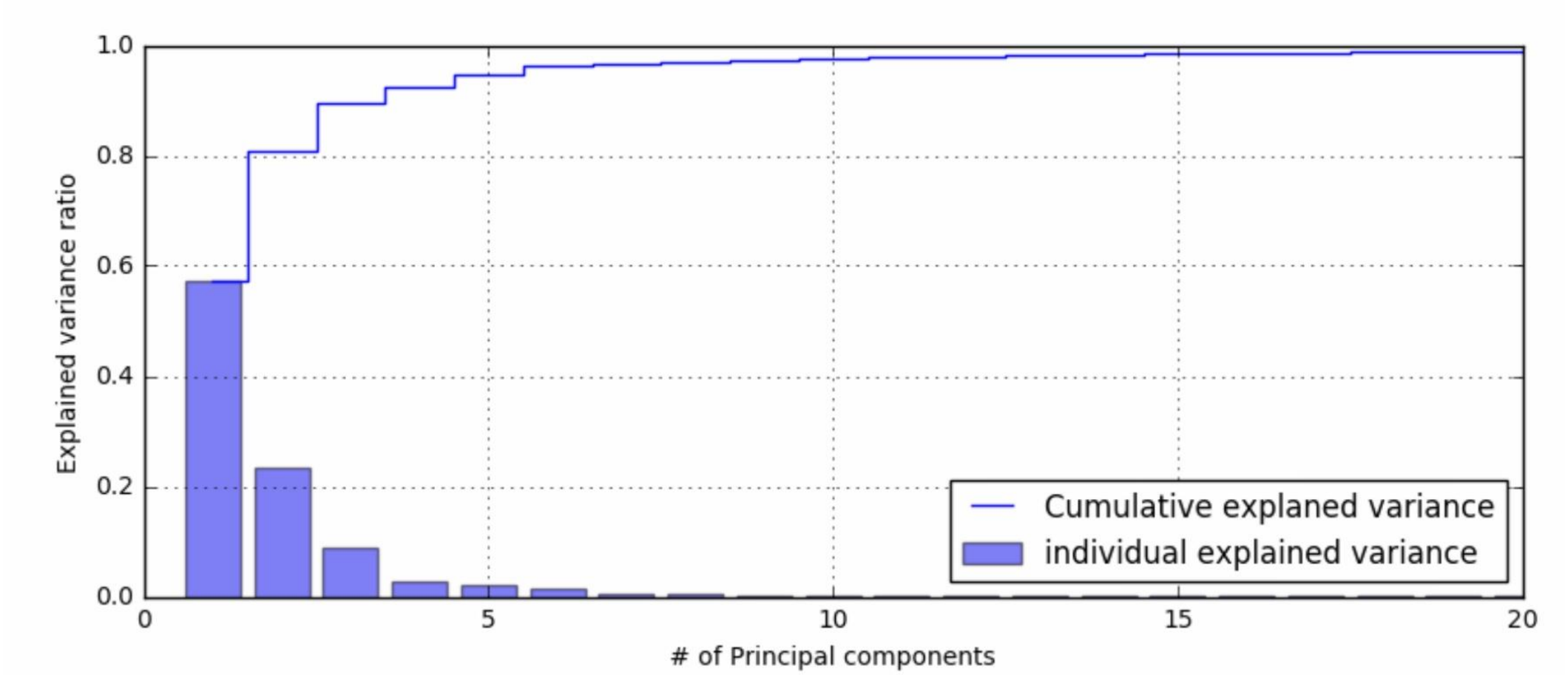
- Example: Semiconductor data (590 features)

Number of components	Percentage of variance	The cumulative percentage
1	59.2	59.2
2	24.1	83.4
3	9.2	92.5
4	2.3	94.8
5	1.5	96.3
6	0.5	96.8
7	0.3	97.1
20	0.08	99.3

Selection of the number of main components



Selection of the number of main components



Practical advice

Speed up supervised learning

- Basic training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \cdots, (x^{(m)}, y^{(m)})\}$
- Extracting entries: $\{x^{(1)}, x^{(2)}, x^{(3)}, \cdots, x^{(m)}\}, \quad x^{(i)} \in \mathbb{R}^{10000}$
- PCA run: $\{z^{(1)}, z^{(2)}, z^{(3)}, \cdots, z^{(m)}\}, \quad z^{(i)} \in \mathbb{R}^{1000}$
- New training series: $\{(z^{(1)}, y^{(1)}), (z^{(2)}, y^{(2)}), (z^{(3)}, y^{(3)}), \cdots, (z^{(m)}, y^{(m)})\}$
- Note: The mapping from the d-dimensional space to the k-dimensional space should be obtained using only the data in the (X_{train}) training set.

PCA application

- Data compression:
  - Reducing the memory consumption required for data storage
  - Increasing the execution speed of the learning algorithm
  - Choosing the number of components: based on the percentage of retained variance

- Data visualization:
  - Choose the number of components: k = 2 or k = 3

The misuse of PCA: dealing with overfitting

- Incorrect method:
  - Using z(i) instead of x(i) reduces the number of features from n to k.
  - As a result, having fewer features reduces the probability of overfitting.
- Correct method:
  - PCA does not use information about the output during dimensionality reduction.
  - Use regularization to deal with overfitting.

$$\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right)^2 + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

The last word

- Designing a machine learning system:
  - Create training set as $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$
  - Using PCA to reduce the dimensions of the $x^{(i)}$s and obtain the $z^{(i)}$s
  - Running the training phase on $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$
  - Hypothesis testing using the test set: mapping $x^{(i)}_{test}$ to $z^{(i)}_{test}$ and calculating $h_\theta(z)$ for each of the test set data.
- An important question: What if we do the above process without using PCA?
  - Always do the above process first without using PCA.
  - If you don't get the desired answer, then experiment with using PCA.

Applications: compression

Training Set

- Training set:
  - 5000 face image (gray)
- Image dimension:
  - 32 * 32 pixel
- Number of features:
  - 1024 features

PCA run with k = 100

- The first 36 components

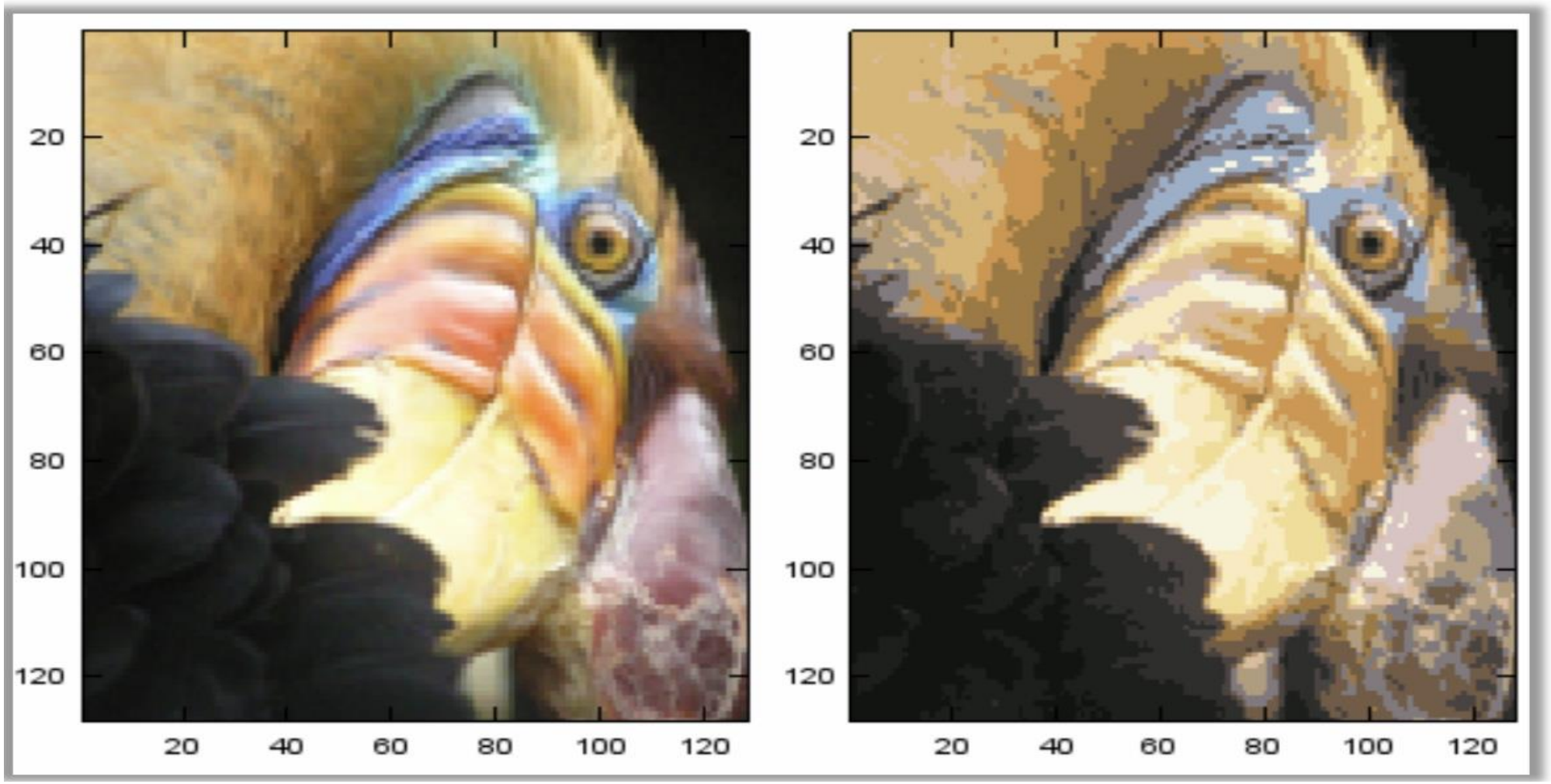
Reconstructed images

Initial images

reconstructed images

Applications: data visualization

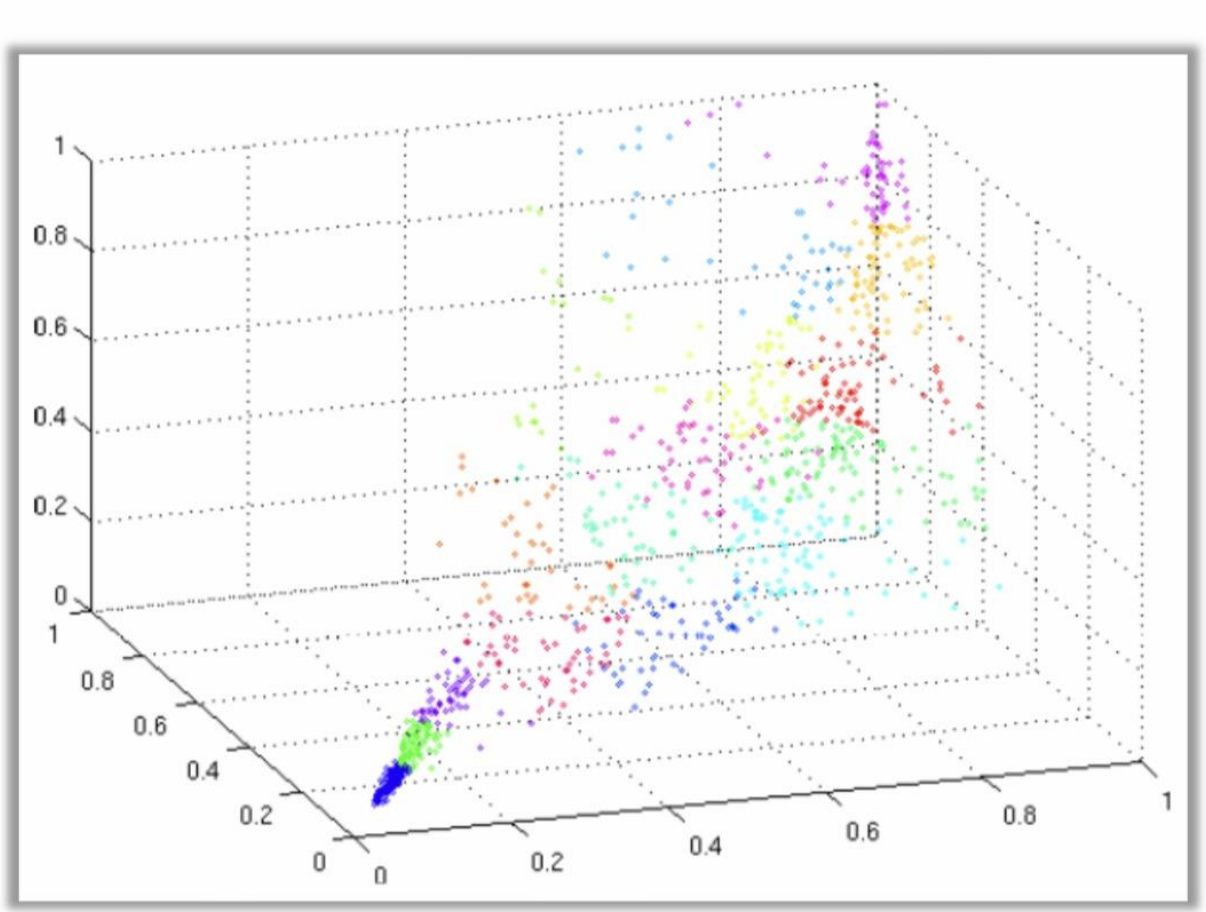
Compression using K-means



The initial image contains thousands of colors (128 * 128)

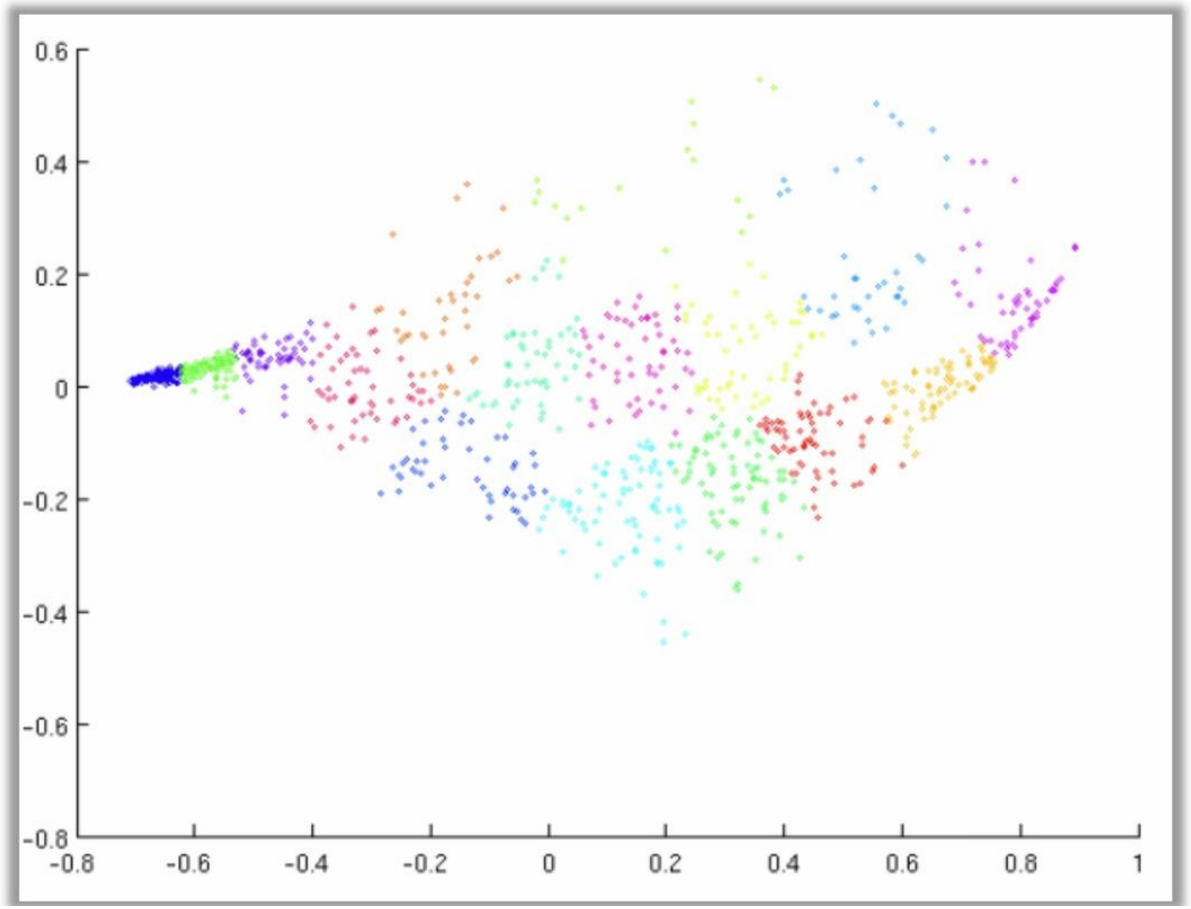
The compressed image contains only 16 colors

Pixels clustering



Pixel clustering by K-means algorithm

The same data in two-dimensional space



It is easier to understand and interpret data in two-dimensional space.

Breakdown of single values

Breakdown of single values

- Motivation:
  - Data simplification
  - Removal of noise and redundancy
  - Improve algorithm results
- Example applications:
  - Information search and retrieval (latent semantic indexing)
  - Recommender systems

Breakdown of single values

- Breakdown of single values:

$$Data_{m\times n} = U_{m\times m}\Sigma_{m\times n}V^{T}_{n\times n}$$

Single value matrix

- Single value matrix:
  - A diagonal matrix in which the individual values are ordered in descending order.
  - Single values from an index such as r onwards have zero value.
  - The singular values of the square root are the eigenvalues of the matrix Data × Data ^{T}.

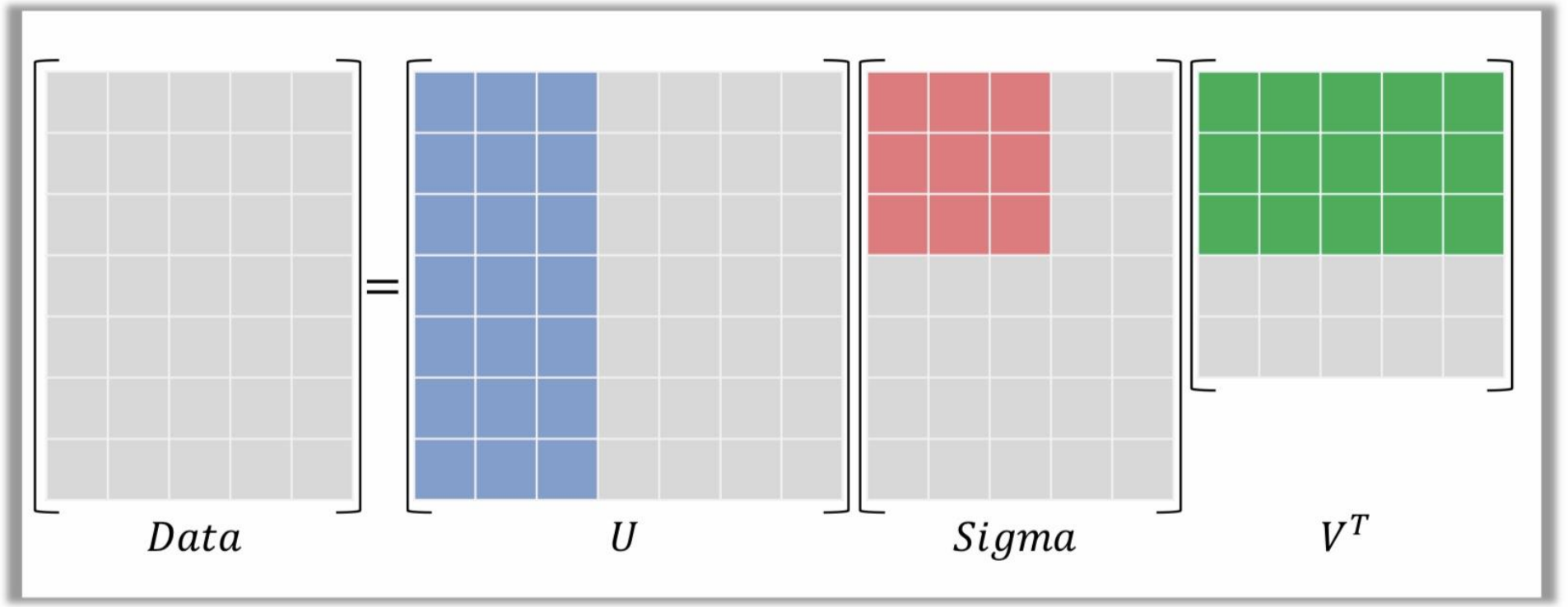
Single value matrix: example

```
>> Data = [1 1 1 0 0; 2 2 2 0 0; 1 1 1 0 0; 5 5 5 0 0;
          1 1 0 2 2; 0 0 0 3 3; 0 0 0 1 1];
>> [U, Sigma, VT] = svd(Data);
>> Sigma
Sigma =
Diagonal Matrix
 9.7214e+000            0            0            0            0
           0  5.2940e+000            0            0            0
           0            0  6.8423e-001            0            0
           0            0            0  9.0541e-016            0
           0            0            0            0  9.3837e-032
           0            0            0            0            0
           0            0            0            0            0
>>
```

$$Data_{m\times n} \approx U_{m\times 3}\Sigma_{3\times 3}V_{3\times n}$$

Single value matrix: example

$$Data_{m\times n} \approx U_{m\times 3}\Sigma_{3\times 3}V_{3\times n}$$



Single value matrix: example (k = 1)

```
>> Sigma1x1 = Sigma(1:1, 1:1)
Sigma1x1 = 9.72140
```

```
>> Data_approx = U(:, 1:1) * Sigma1x1 * VT'(1:1, :)
 1.05   1.05   0.98    0.06    0.06
 2.01   2.01   1.96    0.12    0.12
 1.00   1.00   0.98    0.06    0.06
 5.03   5.03   4.90    0.30    0.30
 0.76   0.76   0.73    0.04    0.04
 0.12   0.12   0.12    0.00    0.00
 0.04   0.04  -0.09    0.00    0.00
```

Initial data

1	1	1	0	0
2	2	2	0	0
1	1	1	0	0
5	5	5	0	0
1	1	0	2	2
0	0	0	3	3
0	0	0	1	1

$SSE \approx 28.494$

Single value matrix: example

```
>> Sigma2x2 = Sigma(1:2, 1:2)
Sigma2x2 =
Diagonal Matrix
   9.72140         0
         0   5.29398

>> Data_approx = U(:, 1:2) * Sigma2x2 * VT'(1:2, :)
 1.00   1.00   0.99   -0.00   -0.00
 2.00   2.00   1.98   -0.00   -0.00
 1.00   1.00   0.99   -0.00   -0.00
 5.02   5.02   4.94   -0.00   -0.00
 0.77   0.77   0.00    2.03    2.03
 0.14   0.14  -0.28    2.98    2.98
 0.05   0.05  -0.09    0.99    0.99
```

Initial data

1	1	1	0	0
2	2	2	0	0
1	1	1	0	0
5	5	5	0	0
1	1	0	2	2
0	0	0	3	3
0	0	0	1	1

$SSE \approx 0.46817$

Single value matrix: example

```
>> Sigma3x3 = Sigma(1:3, 1:3)
Sigma3x3 =
Diagonal Matrix
   9.72140         0         0
         0   5.29398         0
         0         0   0.68423

>> Data_approx = U(:, 1:3) * Sigma3x3 * VT'(1:3, :)
 1.    1.    1.    0.    0.
 2.    2.    2.    0.    0.
 1.    1.    1.   -0.   -0.
 5.    5.    5.    0.    0.
 1.    1.    0.    2.    2.
-0.    0.   -0.    3.    3.
-0.    0.   -0.    1.    1.
```

Initial data

1	1	1	0	0
2	2	2	0	0
1	1	1	0	0
5	5	5	0	0
1	1	0	2	2
0	0	0	3	3
0	0	0	1	1

$$SSE \approx 1.2151e-029$$

Determine the number of individual values

- Determining an appropriate number for individual values:
  - Similar to determining the number of principal components
- An empirical method: choosing the smallest k so that:

$$\frac{\sum_{i=1}^{k} s_{ii}^2}{\sum_{i=1}^{n} s_{ii}^2} \geq 0.90$$

$k = 1.\ energy = 0.768$

$k = 2.\ energy = 0.996$

$k = 3.\ energy = 1.000$

9.72	0	0	0	0
0	5.29	0	0	0
0	0	0.68	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Recommender Systems

- Recommender systems:
  - "Amazon" recommends certain products to customers based on their past purchases.
  - "Netflix" recommends certain movies to its customers based on the watched movies.
  - News websites offer their users to read specific news.
- Approach used: group refinement
  - Comparing data about one user with data about other users
  - Finding users with similar tastes and recommending them based on this similarity

Item-based or user-based similarity?

- Based on items:
  - Example: calculating the similarity of two different movies based on the points given by users
- User-based:
  - Example: calculating the similarity between users based on the points given by them to different movies
- which one? Item-based or user-based similarity calculation?
  - It depends on the number of items and the number of users.
  - If the number of users is much greater than the number of items, then item-based similarity calculation is better.

Appendix: eigenvalues and eigenvectors

eigenvalues and eigenvectors

- Consider the product of the square matrix A in the vector x:
- The matrix A, like a function, transforms the vector x into the new vector y.
- eigenvectors : A vector x is a special vector if it is parallel to the vecto Ax:
- Eigenvalue: In the above relation, lambda is the eigenvalue corresponding to the eigenvector x.

$$Ax = y$$

$$Ax = \lambda x$$

eigenvalues

eigenvecto

eigenvalues and eigenvectors

Example: If A is a permutation matrix as follows:

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

in this case:

$$A\begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1 \times \begin{bmatrix} 1 \\ 1 \end{bmatrix} \Rightarrow \lambda = 1$$

orthogonal

$$A\begin{bmatrix} -1 \\ 1 \end{bmatrix} = (-1) \times \begin{bmatrix} -1 \\ 1 \end{bmatrix} \Rightarrow \lambda = -1$$

$$trace(A) = 0 + 0 = 1 + (-1)$$

Note: the sum of eigenvalues is equal to the sum of elements of the main diameter (effect).

Calculation of eigenvalues and eigenvectors

Calculation of eigenvalues:

$$Ax = \lambda x \Rightarrow (A - \lambda I)x = 0$$

Therefore, the matrix A - λx is a singular matrix. (Because the vector is empty.)
as a result:

$$det(A - \lambda I) = 0$$

Example: Calculation of eigenvalues

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

trace(A) det(A)

$$\det(A - \lambda I) = \det\left(\begin{bmatrix} 3-\lambda & 1 \\ 1 & 3-\lambda \end{bmatrix}\right) = \lambda^2 - 6\lambda + 8 = 0 \Rightarrow \lambda = 4,2$$

Calculation of eigenvalues and eigenvectors

Calculation of eigenvectors:

$$A = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} \Rightarrow \lambda_1 = 4, \lambda_2 = 2$$

Example:

$$(A - 4I)x_1 = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} x_1 = 0 \Rightarrow x_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$(A - 2I)x_2 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} x_2 = 0 \Rightarrow x_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

View:

$$Ax = \lambda x \Rightarrow (A + 3I)x = Ax + 3x = \lambda x + 3x = (\lambda + 3)x$$

Matrix decomposition A: diagonalization

- Let S be a matrix whose columns are eigenvectors of matrix A.

$$
\begin{aligned}
AS &= A\begin{bmatrix} | & | & \cdots & | \\ x_1 & x_2 & \cdots & x_n \\ | & | & \cdots & | \end{bmatrix} \\
&= \begin{bmatrix} | & | & \cdots & | \\ \lambda_1 x_1 & \lambda_2 x_2 & \cdots & \lambda_n x_n \\ | & | & \cdots & | \end{bmatrix} \\
&= \begin{bmatrix} | & | & \cdots & | \\ x_1 & x_2 & \cdots & x_n \\ | & | & \cdots & | \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \\
&= S\Lambda
\end{aligned}
$$

$$AS = S\Lambda \Rightarrow S^{-1}AS = \Lambda$$

$$AS = S\Lambda \Rightarrow A = S\Lambda S^{-1}$$

Matrix decomposition A: diagonalization

- Observation: If we raise the matrix A to the power of two, the eigenvalues reach the power of two and the eigenvectors do not change.

$$A = S\Lambda S^{-1} \Rightarrow A^2 = S\Lambda S^{-1}\, S\Lambda S^{-1} = S\Lambda^2 S^{-1}$$

- Generally:

$$A = S\Lambda S^{-1} \Rightarrow A^k = S\Lambda^k S^{-1}$$

Symmetric matrix decomposition

- In a real symmetric matrix:
- The eigenvalues are real.
- The orthogonal

eigenvectors are normal.

$$A = Q\Lambda Q^{-1} = Q\Lambda Q^{T}$$

$$A = \begin{bmatrix} | & | & \cdots & | \\ q_1 & q_2 & \cdots & q_n \\ | & | & \cdots & | \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \begin{bmatrix} - & q_1^T & - \\ - & q_2^T & - \\ \vdots & \vdots & \vdots \\ - & q_n^T & - \end{bmatrix}$$
$$= \lambda_1 q_1 q_1^T + \lambda_2 q_2 q_2^T + \cdots + \lambda_n q_n q_n^T$$

- Observation: Every symmetric linear combination matrix is a set of orthogonal projection matrices.

Symmetric positive definite matrices

- Positive definite matrix: The matrix A is positive definite if for every non-zero vector such as x:

$$x^T Ax > 0$$

- In a symmetric positive definite matrix:
  - The eigenvalues are all positive.
  - The axes are all positive.
  - The determinants are all positive (the determinants of the preceding matrices).

- Example:

$$A = \begin{bmatrix} 5 & 2 \\ 2 & 3 \end{bmatrix} \Rightarrow \lambda = 4 \pm \sqrt{5}, p_1 = 5, p_2 = \frac{11}{5}, \det(A) = 11$$