Machine Learning
PCA & K-Means

Prof. Chang-Chieh Cheng

Information Technology Service Center

National Chiao Tung University

Decision in `scikit-learn`

- **`sklearn.tree.DecisionTreeClassifier`**
  - https://goo.gl/CiSEFL

- See the example:
  - https://github.com/jameschengcs/ml/blob/master/dts.py

KD Tree in scikit-learn

- **sklearn.neighbors.KDTree**
  - https://goo.gl/2nurtQ
- An example of querying for neighbors within a given radius

```
import numpy as np
from sklearn.neighbors import KDTree

np.random.seed(0)
X = np.random.random((10, 3))  # 10 points in 3 dimensions
tree = KDTree(X, leaf_size=2)
print(tree.query_radius([X[0]], r=0.3, count_only=True))
ind, dist = tree.query_radius( [X[0]],
                               r=0.3,
                               count_only = False,
                               return_distance = True)
print(ind)
print(dist)
```

KD Tree in TensorFlow

- Example:
  - https://goo.gl/YcVx8L

KD Tree and Curse of Dimensionality

- KD tree suffers from curse of dimensionality
  - The number of features is high ➔ high dimension
  - The KD tree becomes higher
  - The search time also be increased

Ball Tree

- A ball tree is a space partitioning data structure for organizing points in a multi-dimensional space.
- It partitions data points into a nested set of **hyperspheres** known as "balls".
- It is useful for nearest neighbor search.
- References:
  - Omohundro, Stephen M. "Five Balltree Construction Algorithms", 1989.
  - M. Dolatshah, et al, "Ball*-tree: Efficient spatial indexing for constrained nearest-neighbor search in metric spaces," *arXiv*, Nov. 2015.

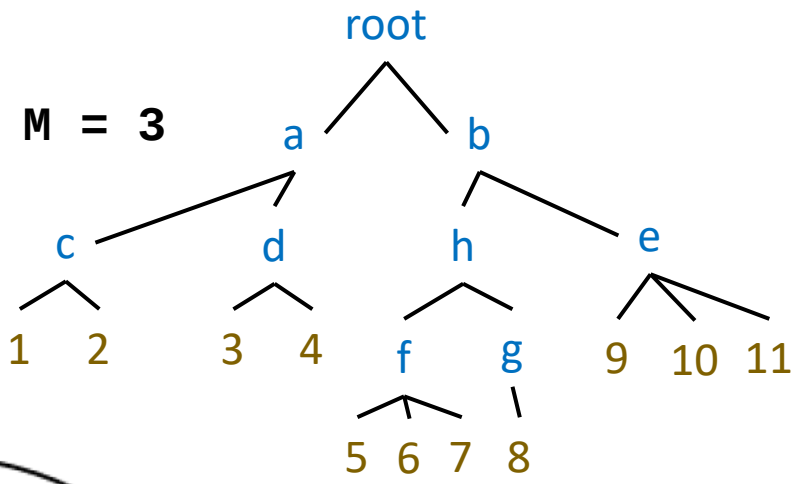
Ball Tree

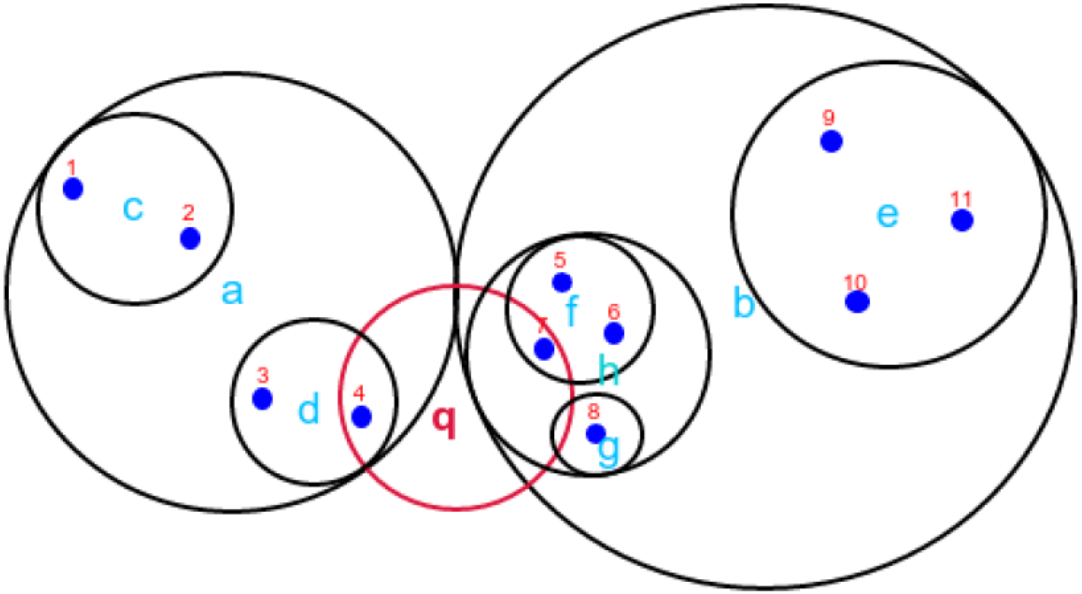
- Construction Algorithm

```
function construct_balltree is
      input:
          D, an array of data points
      output:
          B, the root of a constructed ball tree
      if M points remains then
          create a leaf B containing the M points in D
          return B
      else
          let c be the dimension of greatest spread (PCA)
          let p be the central point selected considering c
          let L,R be the sets of points lying to the left and
              right of the median along dimension c
          create B with two children:
              B.pivot = p
              B.child1 = construct_balltree(L),
              B.child2 = construct_balltree(R),
              let B.radius be maximum distance from p among children
          return B
      end if
  end function
```

Ball Tree

- Example: 11 data instances





Figuring by: M. Dolatshah, et al, "Ball*-tree: Efficient spatial indexing for constrained nearest-neighbor search in metric spaces," *arXiv*, Nov. 2015.

Ball Tree

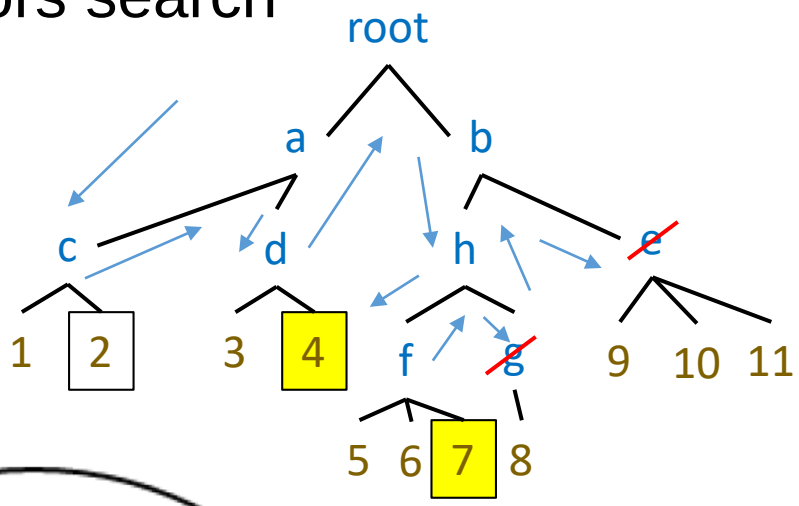
- Nearest neighbor search

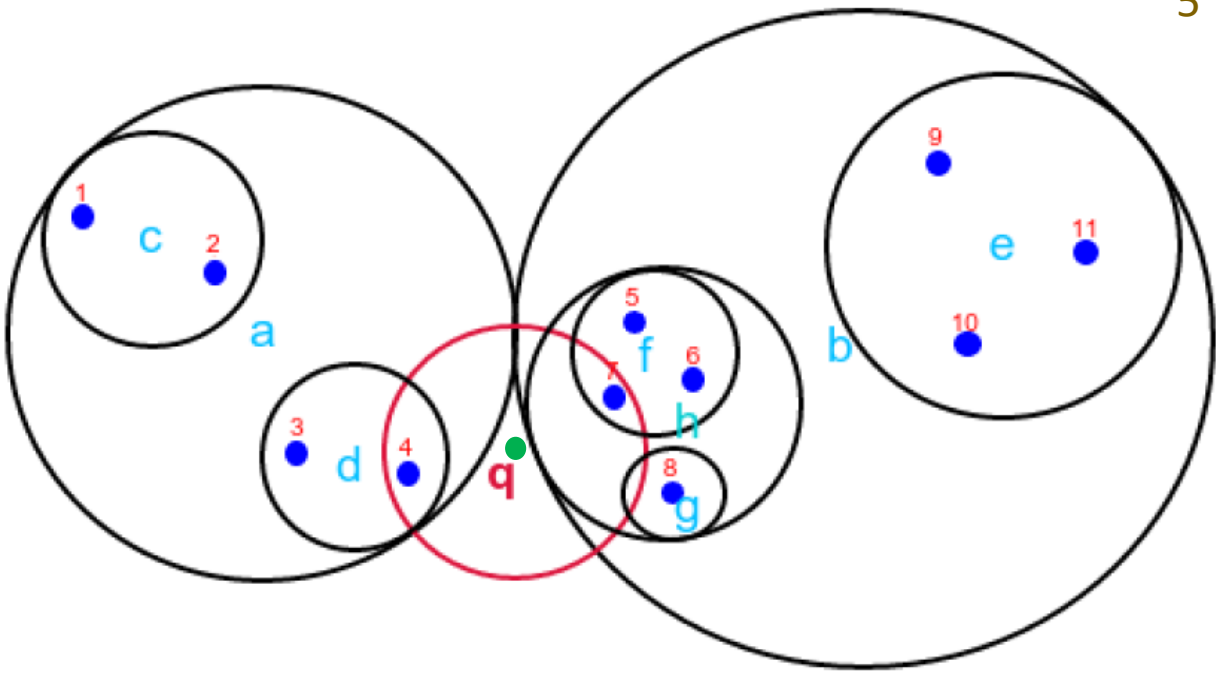
```
function knn_search is
      input:
          t, the target point for the query
          k, the number of nearest neighbors of t to search for
          Q, max-first priority queue containing at most k points
          B, a node, or ball, in the tree
      output:
          Q, containing the k nearest neighbors from within B
      if distance(t, B.pivot) - B.radius ≥ distance(t, Q.first) then
          return Q unchanged
      else if B is a leaf node then
          for each point p in B do
              if distance(t, p) < distance(t, Q.first) then
                  add p to Q
                  if size(Q) > k then
                      remove the furthest neighbor from Q
      else
          let child1 be the child node closest to t
          let child2 be the child node furthest from t
          knn_search(t, k, Q, child1)
          knn_search(t, k, Q, child2)
```

∞ is Q is empty

Ball Tree

- Example: Two nearest neighbors search



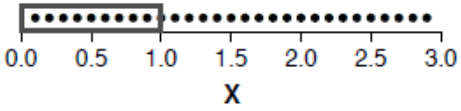


Figuring by: M. Dolatshah, et al, "Ball*-tree: Efficient spatial indexing for constrained nearest-neighbor search in metric spaces," *arXiv*, Nov. 2015.

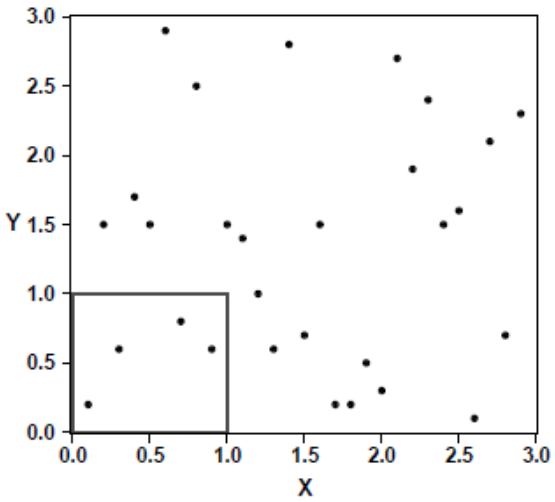
Feature Selection

- The curse of dimensionality
  - $density = k^{\frac{1}{m}}$
    - *k* = the number of data instances
    - *m* = the number of dimensions

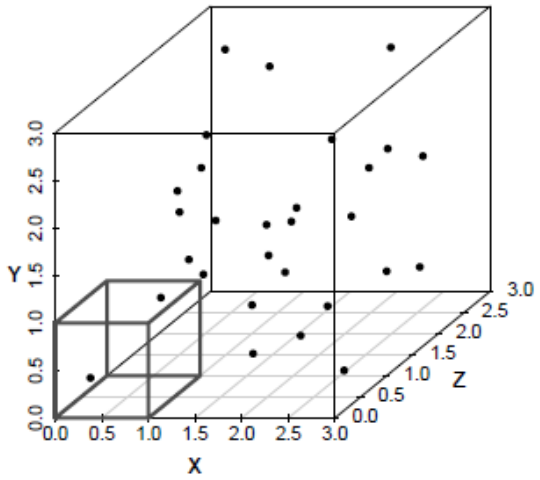
$$density = 10^{\frac{1}{1}} = 10$$



$$density = 4^{\frac{1}{2}} = 2$$



$$density = 2^{\frac{1}{3}} = 1.2599$$



Figuring by: John D. Kelleher, et al, "Fundamentals of Machine Learning for Predictive Data Analytics - Algorithms, Worked Examples, and Case Studies," *MIT Press*, 2015.

- if we fix *k* = 10➔ 2D:

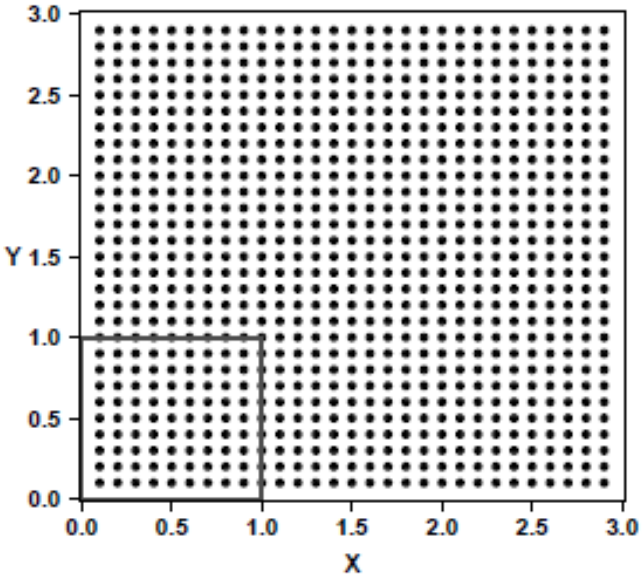
$$density = 10^{\frac{1}{2}} = 3.1623$$

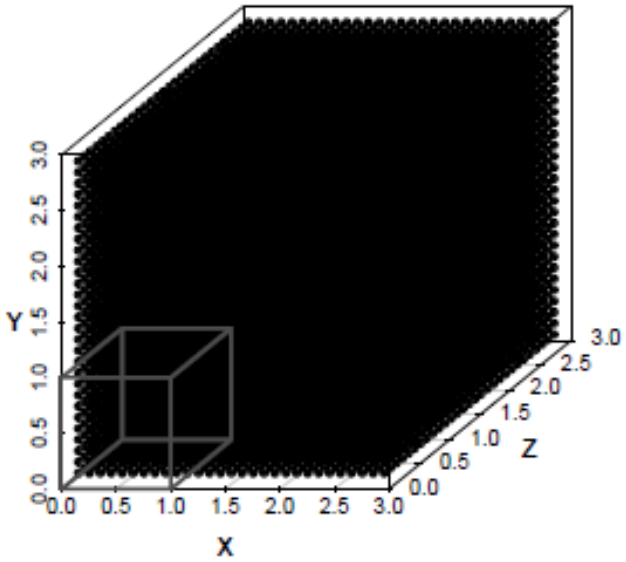
3D:

$$density = 10^{\frac{1}{3}} = 2.1544$$

Feature Selection

- The curse of dimensionality
  - if we fix *density* = 10
    - 1D: $k = 10$
    - 2D: $k^{\frac{1}{2}} = 10 \rightarrow k = 100$
    - 3D: $k^{\frac{1}{3}} = 10 \rightarrow k = 1000$





Figuring by: John D. Kelleher, et al, "Fundamentals of Machine Learning for Predictive Data Analytics - Algorithms, Worked Examples, and Case Studies," *MIT Press*, 2015.

Linear Algebra: Matrix

- Each data instance has *n* features
  - $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$,
  - $\mathbf{x}^{\mathrm{T}} = [x_1 \quad x_2 \quad \cdots \quad x_n]$
- *m* data instances➔ $m \times n$ matrix
  - $\boldsymbol{A} = \begin{bmatrix} \mathbf{x_1}^{\mathrm{T}} \\ \mathbf{x_2}^{\mathrm{T}} \\ \vdots \\ \mathbf{x_m}^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & & \ddots & \\ x_{m1} & x_{m2} & & x_{mn} \end{bmatrix}$

Linear Algebra: Matrix

- M is a symmetric matrix if $M^{\mathrm{T}} = M$
  - $A^{\mathrm{T}}A$ is a symmetric matrix
    - Because $(A^{\mathrm{T}}A)^{\mathrm{T}} = A^{\mathrm{T}}A$

$$A = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & & \ddots & \\ x_{m1} & x_{m2} & & x_{mn} \end{bmatrix}$$

$$A^{\mathrm{T}} = \begin{bmatrix} x_{11} & x_{21} & \cdots & x_{m1} \\ x_{12} & x_{22} & \cdots & x_{m2} \\ \vdots & & \ddots & \\ x_{1n} & x_{2n} & & x_{mn} \end{bmatrix}$$

$$A^{\mathrm{T}}A = \begin{bmatrix} x_{11} & x_{21} & \cdots & x_{m1} \\ x_{12} & x_{22} & \cdots & x_{m2} \\ \vdots & & \ddots & \\ x_{1n} & x_{2n} & & x_{mn} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & & \ddots & \\ x_{m1} & x_{m2} & & x_{mn} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \\ a_{n1} & a_{n2} & & a_{nn} \end{bmatrix},$$

where $a_{ij} = x_{1i}x_{1j} + x_{2i}x_{2j} + \cdots + x_{mi}x_{mj}$

$$= x_{1j}x_{1i} + x_{2j}x_{2i} + \cdots + x_{mj}x_{mi}$$

$$= a_{ji}$$

Linear Algebra: Matrix

- Identity matrix, I

$$I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & & \ddots & \\ 0 & 0 & & 1 \end{bmatrix}$$

  - If A is an $n \times n$ matrix
    - $AI = IA = A$
    - $A^{-1}A = AA^{-1} = I$ ➔ A^{-1} is A's inverse

  - For example:

$$\boldsymbol{A} = \begin{bmatrix} 2 & 1 & 1 \\ 3 & 2 & 1 \\ 2 & 1 & 2 \end{bmatrix} \qquad A^{-1} = \begin{bmatrix} 3 & -1 & -1 \\ -4 & 2 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Linear Algebra: Matrix

- U is an unitary matrix if $U^*U = UU^* = I$
  - *: conjugate transpose
    - if $\boldsymbol{A} = \begin{bmatrix} 1 & -2-3i \\ 1+4i & 5i \end{bmatrix}, \boldsymbol{A}^* = \begin{bmatrix} 1 & 1-4i \\ -2+3i & -5i \end{bmatrix}$
  - $\boldsymbol{U} = \begin{bmatrix} 0.5+0.5i & 0.5-0.5i \\ 0.5-0.5i & 0.5+0.5i \end{bmatrix}, \boldsymbol{U}^* = \begin{bmatrix} 0.5-0.5i & 0.5+0.5i \\ 0.5+0.5i & 0.5-0.5i \end{bmatrix}$
- Q is an orthogonal matrix if $Q^{\mathrm{T}}Q = QQ^{\mathrm{T}} = I$ and Q is a square matrix with real entries
  - For example
    - $\boldsymbol{Q} = \begin{bmatrix} \cos x & \sin x \\ -\sin x & \cos x \end{bmatrix}, \boldsymbol{Q}^{\mathrm{T}} = \begin{bmatrix} \cos x & -\sin x \\ \sin x & \cos x \end{bmatrix}$
    - Because $\cos x \cos x + \sin x \sin x = 1$

Linear Algebra: Matrix

- $\boldsymbol{A}$ is an $m \times n$ matrix
  - Column space: $\boldsymbol{A}\mathbf{x}$, where $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$
    - a space of $m \times 1$ matrices
  - Row space: $\mathbf{x}\boldsymbol{A}$, where $\mathbf{x} = [x_1, x_2, \ldots, x_m]$
    - a space of $1 \times n$ matrices

Linear Algebra: Eigenvalues & Eigenvetcors

- $\boldsymbol{A}$ is an $m \times n$ matrix
  - An eigenvector of A is a non-zero vector $\mathbf{v}$ such that

$$\boldsymbol{A}\mathbf{v} = \lambda\mathbf{v}$$

or

$$\mathbf{v}^{\mathrm{T}}\boldsymbol{A} = \lambda\mathbf{v}^{\mathrm{T}}$$

  - The scalar λ is the eigenvalue corresponding to $\mathbf{v}$
  - Given a non-zero real number s, $s\mathbf{v}$ and $\mathbf{v}$ have the same eigenvalue
    - $\boldsymbol{A}\mathbf{v} = \lambda\mathbf{v}$
    - $\boldsymbol{A}(s\mathbf{v}) = \lambda(s\mathbf{v})$
  - We often let every eigenvector is normalized (the length is one)

Linear Algebra: Eigenvalues & Eigenvetcors

- Linearly independent
  - A set of n vectors, $V = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$
  - V is linearly independent if
    $a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \ldots + a_n\mathbf{v}_n$ = zero vector, can only be satisfied by all a_i are zeros

- Suppose that A has n linearly independent eigenvectors,
  - $\boldsymbol{A}$ is an $m \times n$ matrix
  - Q is an $n \times n$ matrix = $[\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n]$
  - $\boldsymbol{\Lambda}$ is an $n \times n$ diagonal matrix with n eigenvalue
  - $AQ = Q\boldsymbol{\Lambda}$

Linear Algebra: Eigenvalues & Eigenvetcors

- Every **real symmetric matrix** can be decomposed as follows
  - $\boldsymbol{A}$ is an $n \times n$ real symmetric matrix
  - $A = Q \Lambda Q^{\mathrm{T}}$
    - Q contains n orthogonal eigenvectors
      - $Q Q^{\mathrm{T}} = I$
    - Λ is an $n \times n$ diagonal matrix contains n eigenvalues

Linear Algebra: Eigenvalues & Eigenvetcors

- Singular value decomposition, SVD
  - Every real matrix has a singular value decomposition
    - A is an $m \times n$ matrix
    - $A = UDV^{\mathrm{T}}$
      - U is an $m \times m$ matrix, $UU^{\mathrm{T}} = U^{\mathrm{T}}U = I$
      - D is an $m \times n$ diagonal matrix
      - V is an $n \times n$ matrix, $VV^{\mathrm{T}} = V^{\mathrm{T}}V = I$
  - But the same is not true of the eigenvalue decomposition.
    - For example, if a matrix is not square, the eigen decomposition is not defined, and we must use a singular value decomposition instead.

Linear Algebra: Eigenvalues & Eigenvetcors

- $\boldsymbol{A}$ is an $m \times n$ real matrix
  - $A = UDV^{\mathrm{T}}$
- $A^{\mathrm{T}}A$ is an $n \times n$ real symmetric matrix
  - $A^{\mathrm{T}}A = (UDV^{\mathrm{T}})^{\mathrm{T}}UDV^{\mathrm{T}} = VDU^{\mathrm{T}}UDV^{\mathrm{T}} = VD^2V^{\mathrm{T}}$
  - $Q = V$
  - $\Lambda = D^2$
- AA^{T} has the same property

Linear Algebra: Eigenvalues & Eigenvetcors

- The center of *m* data instances
  - $\bar{\mathbf{x}} = \frac{1}{m}\sum_{i=1}^{m} \mathbf{x}_i$
- *m* data instances➔ $m \times n$ matrix
  - *m* data instances
  - *n* dimensions (features)
  - $\boldsymbol{A} = \begin{bmatrix} (\mathbf{x_1}-\bar{\mathbf{x}})^{\mathrm{T}} \\ (\mathbf{x_2}-\bar{\mathbf{x}})^{\mathrm{T}} \\ \vdots \\ (\mathbf{x_m}-\bar{\mathbf{x}})^{\mathrm{T}} \end{bmatrix}$
  - $A^{\mathrm{T}}A$ ($n \times n$) is the covariance matrix to describe the data variance of all dimensions (features)

Linear Algebra: Covariance Matrix

- $\boldsymbol{A} = \begin{bmatrix} (\mathbf{x_1} - \bar{\mathbf{x}})^{\mathrm{T}} \\ (\mathbf{x_2} - \bar{\mathbf{x}})^{\mathrm{T}} \\ \vdots \\ (\mathbf{x_m} - \bar{\mathbf{x}})^{\mathrm{T}} \end{bmatrix}$

$$A^{\mathrm{T}}A = \begin{bmatrix} x_{11} - \bar{x}_{11} & x_{21} - \bar{x}_{21} & \cdots & x_{m1} \_ \bar{x}_{m1} \\ x_{12} - \bar{x}_{12} & x_{22} - \bar{x}_{22} & \cdots & x_{m2} \_ \bar{x}_{m2} \\ \vdots & & \ddots & \\ x_{1n} - \bar{x}_{1n} & x_{2n} - \bar{x}_{2n} & & x_{mn} - \bar{x}_{mn} \end{bmatrix} \begin{bmatrix} x_{11} - \bar{x}_{11} & x_{12} - \bar{x}_{12} & \cdots & x_{1n} - \bar{x}_{1n} \\ x_{21} - \bar{x}_{21} & x_{22} - \bar{x}_{22} & \cdots & x_{2n} - \bar{x}_{2n} \\ \vdots & & \ddots & \\ x_{m1} \_ \bar{x}_{m1} & x_{m2} \_ \bar{x}_{m2} & & x_{mn} - \bar{x}_{mn} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \\ a_{n1} & a_{n2} & & a_{nn} \end{bmatrix},$$

where $a_{ij} = a_{ji}$

$$= (x_{1i} - \bar{x}_{1i})(x_{1j} - \bar{x}_{1j}) + (x_{2i} - \bar{x}_{2i})(x_{2j} - \bar{x}_{2j}) + \cdots + (x_{mi} - \bar{x}_{mi})(x_{mj} - \bar{x}_{mi})$$

Variance of ***j*-th** feature of the **first** data instance

Variance of ***j*-th** feature of the **second** data instance

Variance of ***i*-th** feature of the **first** data instance

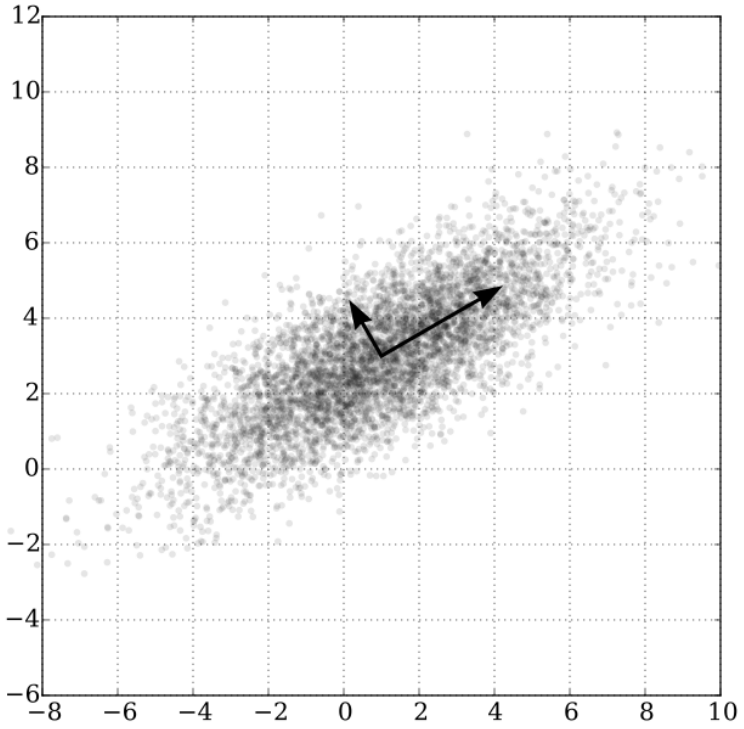
Variance of ***i*-th** feature of the **second** data instance

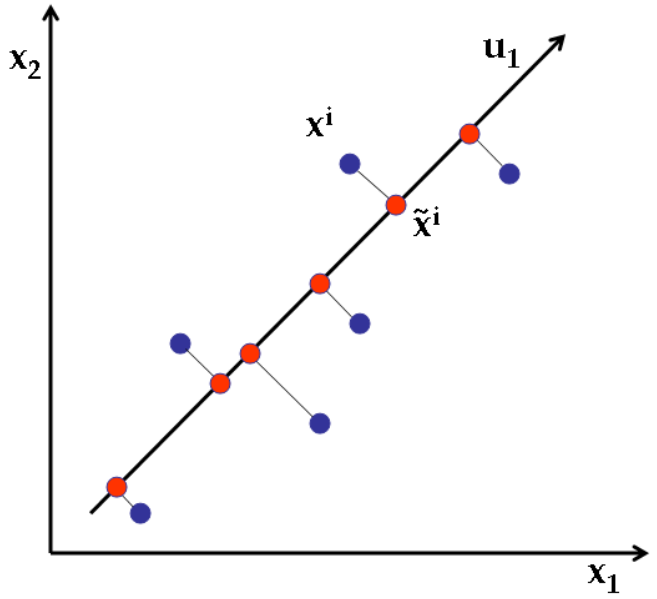
Linear Algebra: Covariance Matrix

- Covariance matrix accounts variability in the dataset.
- Covariance matrix can summarize how much information in the data
- If all same values as its observations, then the variance is 0
- Covariance matrix also can tell us that how to transform the dataset and project that to an axis such that the projection area is maximum

PCA

- Principal component analysis, PCA
  - The covariance matrix of a dataset ➔ PCA
  - The goal of PCA is to find projection directions along which the variance of the projected data points is maximum





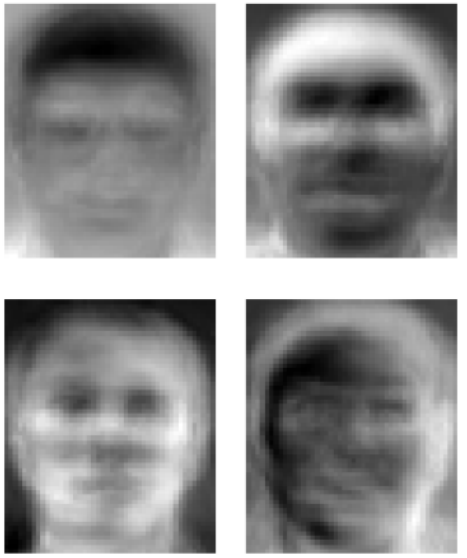
Figuring by: Wikipedia, https://en.wikipedia.org/wiki/Principal_component_analysis

PCA

- Principal components:
  - The eigenvectors ordered with eigenvalues
  - $A^{\mathrm{T}}A = Q\Lambda Q^{\mathrm{T}}$
    - where $\Lambda = \begin{bmatrix} \lambda_1 & & 0 \\ & \lambda_2 & \\ 0 & & \ddots \\ & & & \lambda_n \end{bmatrix}, \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$
    - $Q = [\mathbf{v_1} \quad \mathbf{v_2} \quad \cdots \quad \mathbf{v_n}]$
  - The data can be represented by in a space with the basis of Q
    - $\mathbf{y}^{\mathrm{T}} = (\mathbf{x} - \bar{\mathbf{x}})^{\mathrm{T}}Q$
    - The axis $\mathbf{v_1}$ is more importance than $\mathbf{v_2}$
    - Choose $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_k}$, where λ_k larger than a threshold, we call these axes are principal components

PCA Example: Face Recognition

- L. Sirovich; M. Kirby (1987). "Low-dimensional procedure for the characterization of human faces". *Journal of the Optical Society of America A*. 4 (3): 519–524.
- Eigenfaces
  - Eigenvectors of a training set of *m* face images
  - Each image: $r \times c$ pixels ➔ let $n = r \times c$ ➔a vector of *n* intensities
  - The average of *m* face images➔$\bar{\mathbf{x}}$
  - *m* face images - $\bar{\mathbf{x}}$ ➔ $A = m \times n$ matrix
  - Finding the eigenvectors of $A^{\mathrm{T}}A$
  - Each eigenvector consists of *n* intensities
  - eigenvectors ➔ eigenfaces



AT&T Laboratories Cambridge

PCA Example: Face Recognition

- Choose the principal components

$$\bar{\lambda} = (\lambda_1 + \lambda_n + \cdots + \lambda_n)$$

  - Given a threshold *t*, we choose *k* principal components such that

$$\frac{(\lambda_1+\lambda_n+\cdots+\lambda_k)}{\bar{\lambda}} > t$$

  - Therefore, each training face image can be represented by
    - $\mathbf{y}^{\mathrm{T}} = (\mathbf{x} - \bar{\mathbf{x}})^{\mathrm{T}} Q$
    - ➔$\mathbf{y}^{\mathbf{T}} Q^{\mathbf{T}} = (\mathbf{x} - \bar{\mathbf{x}})^{\mathrm{T}}$
    - ➔$\mathbf{x} = Q\boldsymbol{y} + \bar{\mathbf{x}}$
    - $y_1\times$ 1st eigenface + $y_2\times$ 2nd eigenface + … + $y_k\times$ kth eigenface

PCA Example: Face Recognition

- Face recognition
  - input face: $\mathbf{z}$
  - $\mathbf{w^T} = (\mathbf{z} - \bar{\mathbf{x}})^{\mathrm{T}} Q$
  - Find the nearest neighbour of $\mathbf{w^T}$ from the transformed training set (all $\mathbf{y}$)

PCA Example: Feature Analysis

- Reference:
  - Jeff Jauregui "Principal component analysis with linear algebra", 2012.
- Sibley's Bird Database of North American birds
- 100 bird species. ➔ $m = 100$
- In studying **the size of a bird**, each specie should be observed by three features: **length**, **wingspan**, and **weight**. ➔ $n = 3$



PCA Example: Feature Analysis

- $A^{\mathrm{T}}A = \begin{bmatrix} 91.43 & 171.92 & 297.99 \\ 545.21 & 373.92 & 545.21 \\ 297.99 & 171.92 & 1297.26 \end{bmatrix}$

- ➔$\lambda_1 = 1626.52, \lambda_2 = 128.99, \lambda_3 = 7.10$

- ➔$\mathbf{v}_1 = \begin{bmatrix} 0.22 \\ 0.41 \\ 0.88 \end{bmatrix}, \mathbf{v_2} = \begin{bmatrix} 0.25 \\ 0.85 \\ -0.46 \end{bmatrix}, \mathbf{v_3} = \begin{bmatrix} 0.97 \\ -0.32 \\ -0.08 \end{bmatrix}$
- Which of the feature is most significant to distinguish Sibley birds?

PCA Example: Feature Analysis

- The third entry, weight, of $\mathbf{v}_1$ is the largest
  - Weight is the most significant.
  - Wingspan is the next most important factor.
- $\mathbf{v}_2$ is also telling us something
  - Some birds are large size with small wingspan and large weight ➔ stoutness birds

PCA

- PCA in sklearn

```
import numpy as np
from sklearn import datasets
from sklearn import decomposition

np.random.seed(5)
iris = datasets.load_iris()
X = iris.data
y = iris.target

pca = decomposition.PCA(n_components=3)
pca.fit(X)
print( pca.components_ )
    #[[ 0.36158968 -0.08226889  0.85657211  0.35884393]
    # [ 0.65653988  0.72971237 -0.1757674  -0.07470647]
    # [-0.58099728  0.59641809  0.07252408  0.54906091]]
```

K-Means Clustering

- Clustering *n* data points **X** into *k* disjoint subsets S_i containing n_i data points so as to minimize the sum-of-squares criterion.

$$\mathbf{X} = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}\}, \mathbf{S} = \{\boldsymbol{S_1}, \boldsymbol{S_2}, \ldots, \boldsymbol{S_k}\}$$

$$\underset{\boldsymbol{S}}{\operatorname{argmin}} \sum_{\boldsymbol{i=1}}^{\boldsymbol{k}} \sum_{\boldsymbol{x \in S_i}} \|\boldsymbol{x} - \boldsymbol{\mu_i}\|^{\mathbf{2}}$$

- where $\boldsymbol{\mu_i}$ is the center of $\boldsymbol{S_i}$
- *K*-means clustering is a type of **unsupervised learning**, which is used when data without defined categories.
- References:
  - https://en.wikipedia.org/wiki/K-means_clustering
  - http://mathworld.wolfram.com/K-MeansClusteringAlgorithm.html
  - https://www.datascience.com/blog/k-means-clustering
  - http://www.saedsayad.com/clustering_kmeans.htm

K-Means Clustering

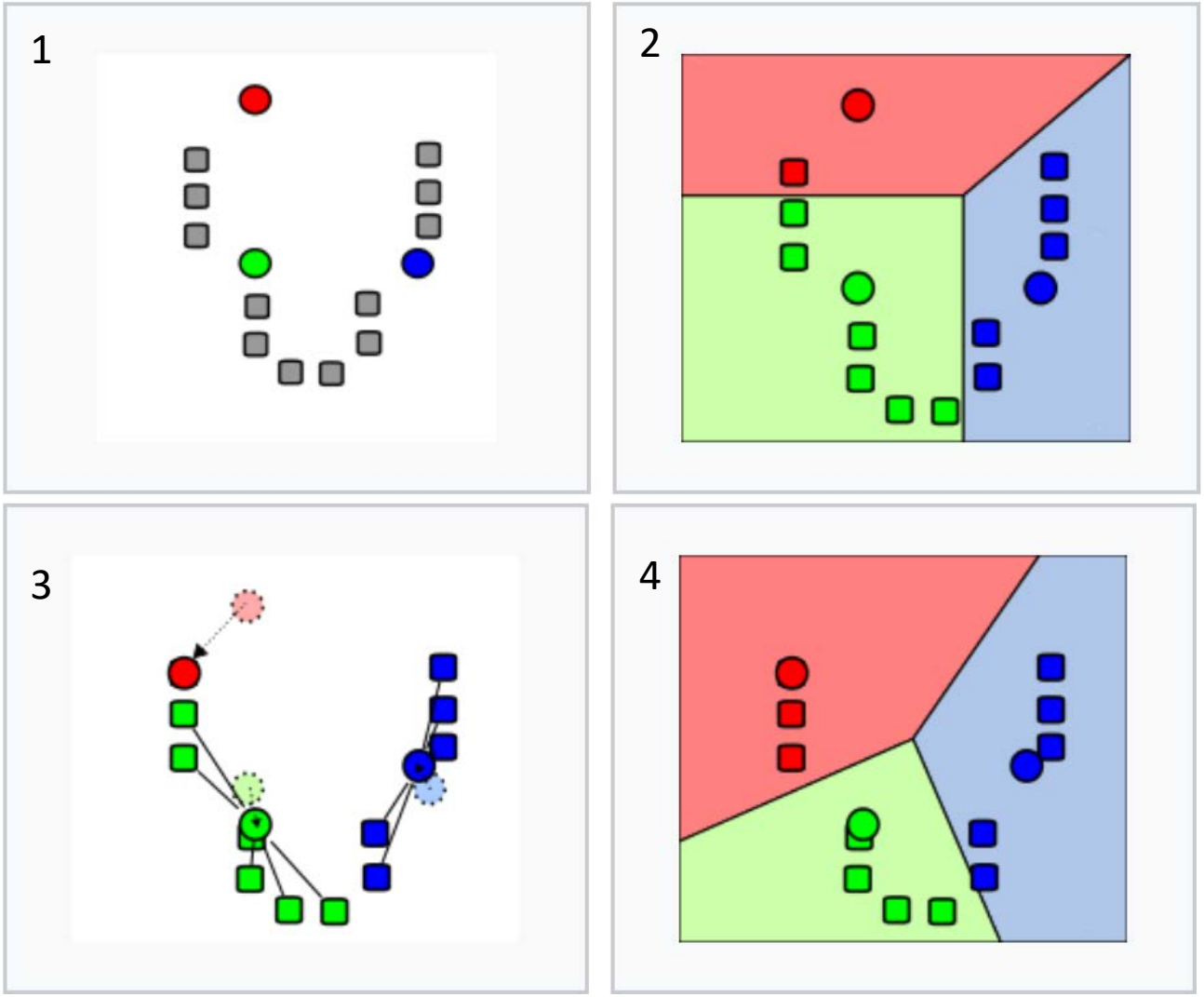
- Standard algorithm

  Input: **X** and *k.*

  1. Select *k* points at random as cluster centers.
     - These *k* points may not ∈ **X**
  2. Assign data instances to their closest cluster center according to the Euclidean distance function.
     - Generating **S**
     - Voronoi diagram
  3. Updating the cluster center by the mean of data instances in each cluster.
  4. Repeat steps 2 and 3 until a stopping criteria is met:
     - No data points change clusters.
     - The sum of the distances is minimized.
     - Some maximum number of iterations is reached.

K-Means Clustering

- Standard algorithm



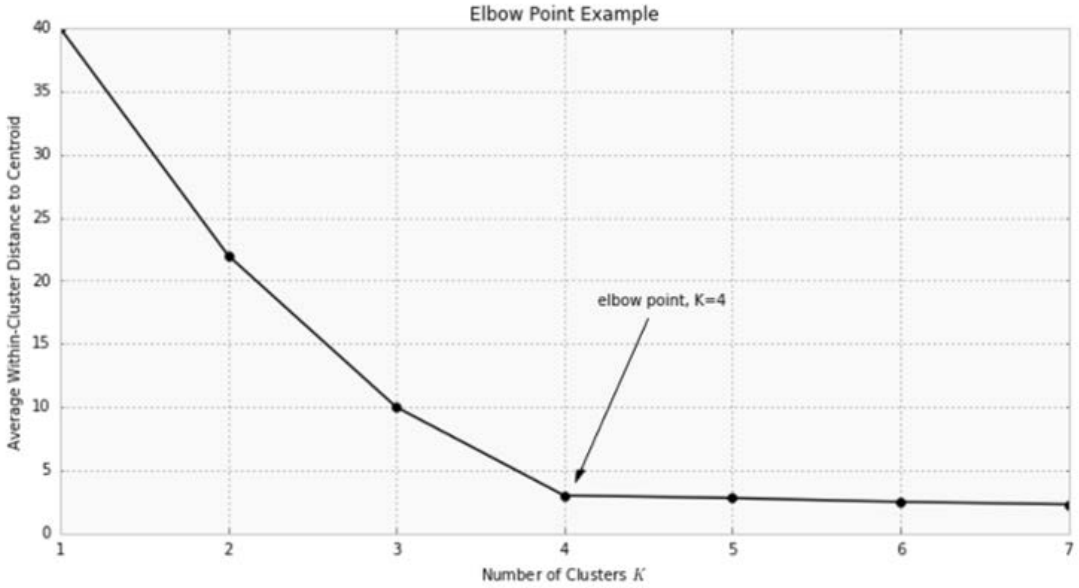
Figuring by: Nathan Landman, Hannah Pang, Christopher Williams, https://brilliant.org/wiki/k-means-clustering/

K-Means Clustering

- How to decide *k*?
  - There is no perfect method for determining exact value of *K.*
  - An approximate algorithm
    - Increasing the *k* will always reduce the distance to data points
    - Calculating

$$\alpha_k = \frac{1}{k}\sum_{i=1}^{k}\sum_{x \in S_i} \|x - \mu_i\|^2$$

    - Select *k* while $\frac{d\alpha_k}{dk}$ less than a small number



Figuring by: Andrea Trevino, https://www.datascience.com/blog/k-means-clustering

K-Means Clustering

- Applications
  - Segment by purchase history
  - Segment by activities on a web-based application
  - Define personas based on interests
  - Detect activity types in motion sensors
  - Group images
  - Separate audio

K-Means Clustering

- KMeans in sklean

```
from sklearn.cluster import KMeans
import numpy as np

X = np.array([[0.5, 2], [1, 4.5], [1, 0.25],
              [4, 2], [4, 4], [4, 0]])

kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
print(kmeans.labels_)                 # [0 0 0 1 1 1]
print(kmeans.cluster_centers_)        #[[0.833 2.25]
                                      # [4.  2. ]]
targets = kmeans.predict([[0, 0], [4, 3]])
print(targets)                        # [0 1]
```