Unsupervised Learning

Andrea De Simone

SISSA INFN Istituto Nazionale di Fisica Nucleare

andrea.desimone@sissa.it

Market Segmentation

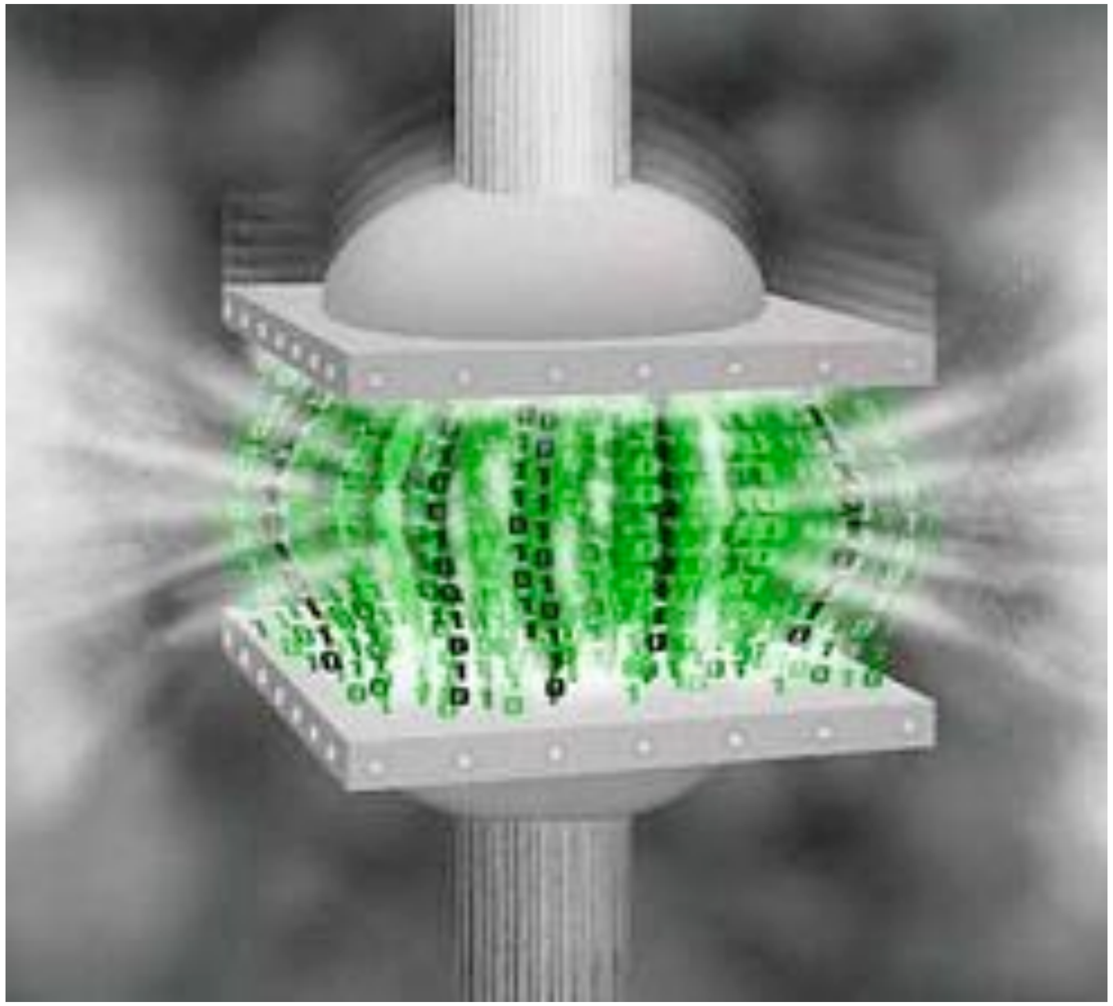
Social Networks

> Invitation

Fraud Detection

Hacker Intrusions

Data compression



> Outline

1. Cluster Analysis

- K-means
- Hierarchical clustering

2. Anomaly detection

- Multi-variate Gaussian model
- Nearest Neighbors

3. Dimensionality Reduction

- Principal Component Analysis

> References

- James, Witten, Hastie, Tibshirani, "*An Introduction to Statistical Learning*", Springer (2013), Chapter 10.

- Hastie, Tibshirani, Friedman, "*The Elements of Statistical Learning*", Springer (2008), Chapter 14.

- Coursera/Udacity courses
  - https://www.coursera.org/learn/machine-learning
  - https://eu.udacity.com/course/machine-learning-unsupervised-learning--ud741

- sklearn tutorial online

> Types of Machine Learning

Supervised Learning

Unsupervised Learning

Reinforcement Learning

> Supervised Learning

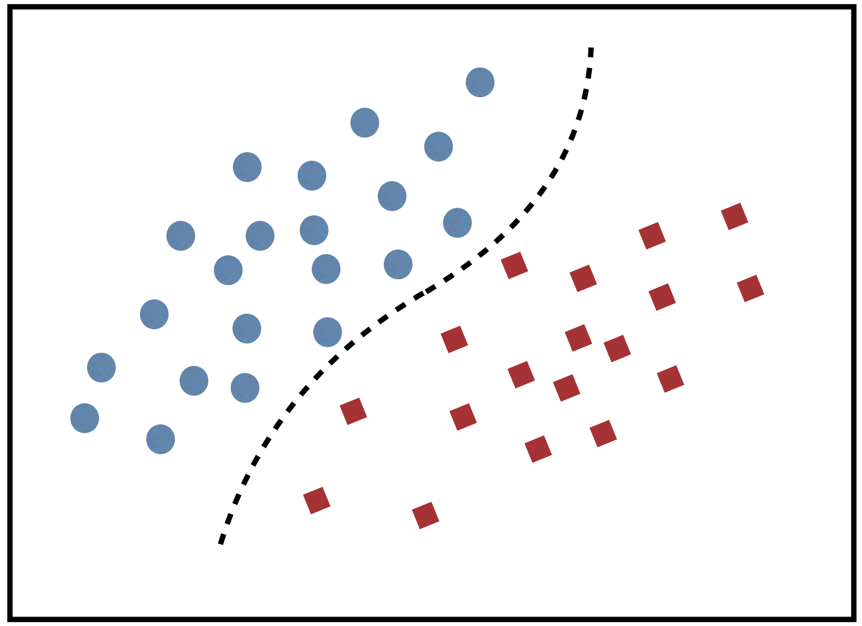
labelled data

$$\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$$

features

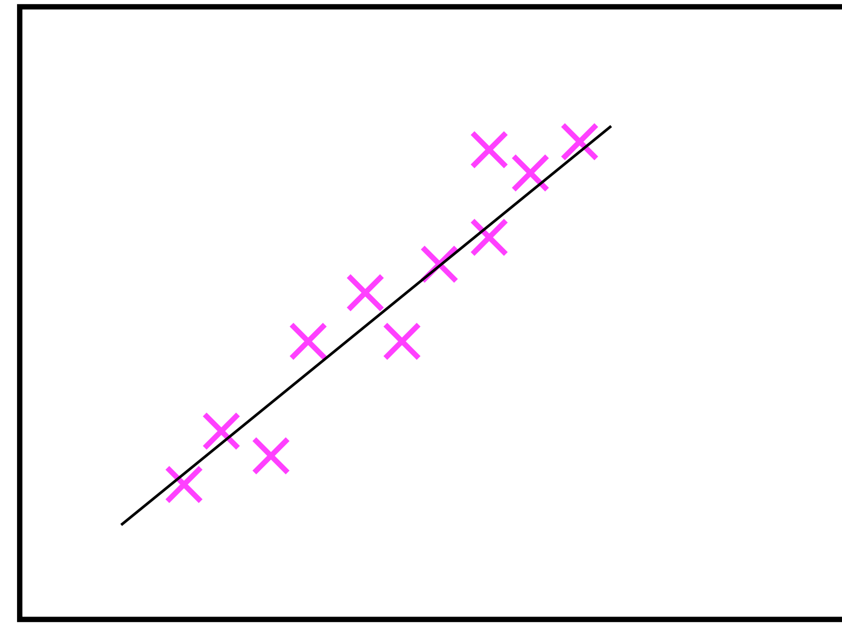
labels

Classification



Logistic Regression
Neural Networks
Decision Trees
Nearest Neighbors
…

Regression



Polynomial Regression
Neural Networks
Support Vector Machines
Nearest Neighbors
…

machine "learns" the model

$$f(x) = y$$

…or the conditional density *P(Y|X)*

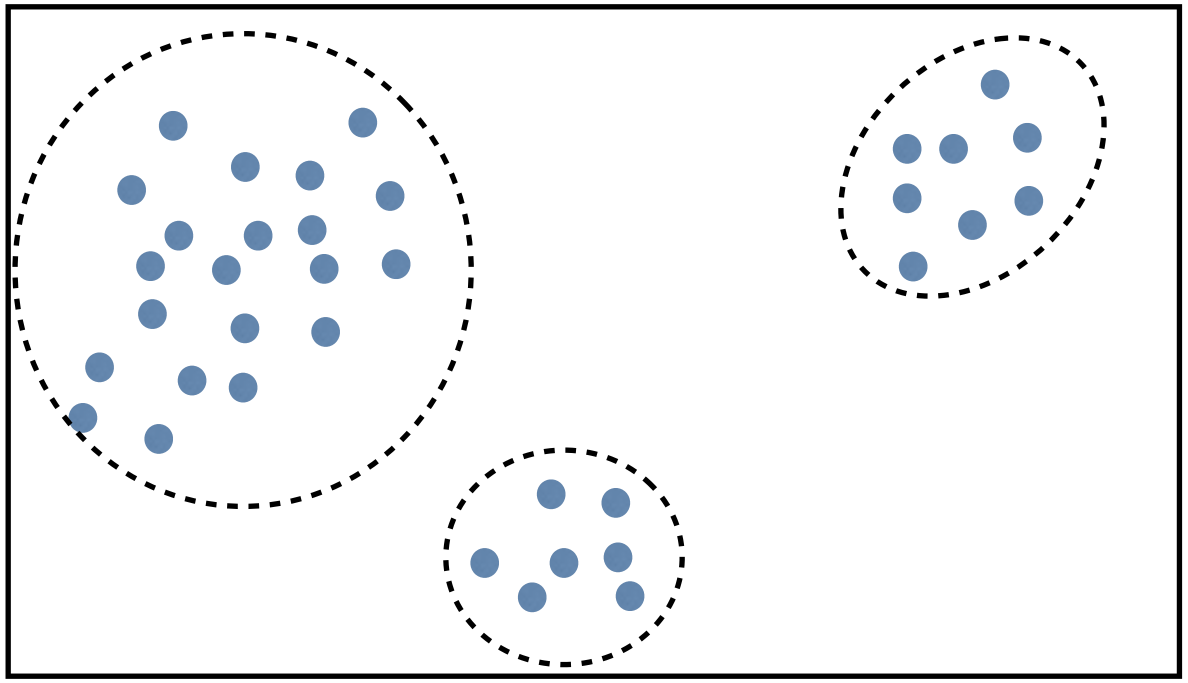
$$P(X,Y) = P(Y|X) \cdot P(X)$$

> Unsupervised Learning

unlabelled data

$\{\boldsymbol{x}_i\}_{i=1}^N$

features



Cluster Analysis
Dimensionality Reduction
Anomaly Detection
…

machine “learns”
patterns, structures, representations, etc.
of the data

…or the properties of the joint pdf *P(X)*

What do your data look like?

$$\{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(N)}, y^{(N)})\}$$

features

labels

SUPERVISED LEARNING

$$\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N)}\}$$

features

UNSUPERVISED LEARNING

What is your task?

organize data by similarity	→	CLUSTER ANALYSIS
find unexpected events (outliers)	→	ANOMALY DETECTION
reduce the dimension of feature space	→	DIMENSIONALITY REDUCTION
…	→	…

> Supervised vs. Unsupervised

Supervised Learning

- optimization objective (loss function)
- performance metrics
- low/medium dimensions of feature space
- interested in estimates of location parameters

Unsupervised Learning

- no loss function → harder than supervised
- data mining without a model
- typically high-dimensional feature space
- interested in more complex properties of data

Unsupervised learning may be used to **pre-process** data for Supervised learning

1
CLUSTER ANALYSIS

Approaches to Clustering

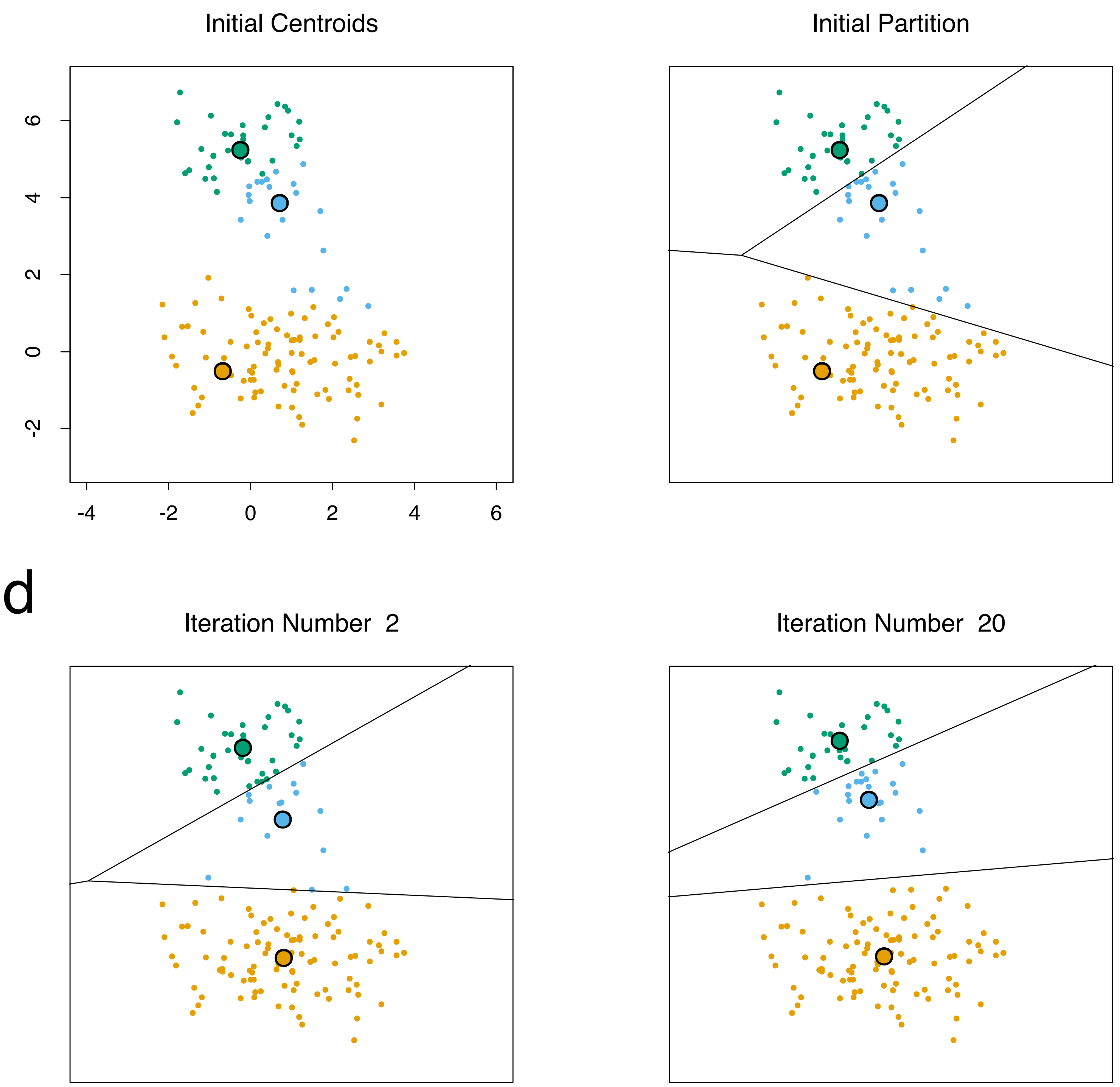
- group objects based on their “similarity” by:
  - maximizing similarity within same cluster
  - minimizing similarity between different clusters

- K-means
  - very popular and simple
  - need to specify number of clusters

- Hierarchical
  - need to specify dissimilarity measure
  - bottom-up (agglomerative)
  - top-down (divisive)

Basic idea

1. initialize cluster centers (centroids)
2. assign points to cluster with smallest distance to its centroid
3. update centroids to mean of cluster points
4. iterate 2. and 3.



[*Voronoi* tassellation]

- **Input:** set of *N* examples (points) $\{x^{(1)}, \ldots, x^{(N)}\}$ $\quad x^{(i)} \in \mathbb{R}^D$
- **Input:** number of clusters *K*

1. randomly initialize *K* cluster centroids $\mu_1, \ldots, \mu_K \in \mathbb{R}^D$

2. repeat {

    for *i=1* **to** *N* *(loop over dataset)*

    - cluster assignment $C(i) = \underset{1 \leq j \leq K}{\operatorname{argmin}} ||x^{(i)} - \mu_j||^2$ (cluster to which $x^{(i)}$ is assigned)

    for *k=1* **to** *K* *(loop over centroids)*

    - cluster partitions $N_k = \sum_{i=1}^{N} I(C(i) = k)$ (s.t. $\sum_{k=1}^{K} |N_k| = N$)
    - move cluster centroids $\mu_k = \frac{1}{|N_k|} \sum_{i \in N_k} x^{(i)}$

   } until assignments do not change

- **Output:** cluster centroids $\mu_1, \ldots, \mu_K$

Optimization objective

- minimize over cluster assignments *C(i)* and cluster centroids μ_k

$$\min_{\{C\},\{\mu\}} L(\{C\},\{\mu\})$$

$$L(\{C\},\{\mu\}) \equiv L(C(1),\ldots,C(N),\mu_1,\ldots,\mu_K) = \sum_{j=1}^{K}\sum_{i\in N_k} ||x^{(i)}-\mu_j||^2$$

- cluster assignment step for fixed $\mu_1,\ldots,\mu_k$

$$\min_{\{C\}} L(\{C\},\{\mu\}) \longrightarrow C(i) = \operatorname*{argmin}_{1\le j\le K} ||x^{(i)}-\mu_j||^2$$

- move centroid step for fixed $C(1),\ldots,C(N)$

$$\min_{\{\mu\}} L(\{C\},\{\mu\}) \longrightarrow \mu_k = \operatorname*{argmin}_{\{\mu\}} \sum_{i\in N_k} ||x^{(i)}-\mu||^2$$

Random Initialization

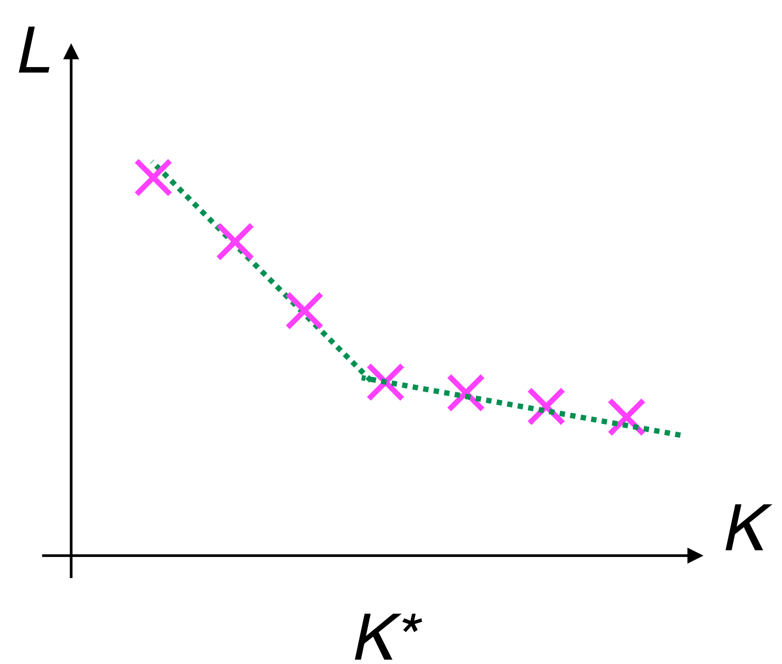
- pick *K* examples randomly $\{x^{(Z_1)}, \ldots, x^{(Z_K)}\}$
- set initial centroids: $\mu_1 = x^{(Z_1)}, \ldots, \mu_K = x^{(Z_K)}$

Problem: different initial conditions may end up with different clusters (different local minima)

Solution: run K-means for many different random initializations and choose clustering minimizing loss *L*

Choose K

- what is the best value of K?
- Loss function decreases with K
- observation: let K^* be the "true" number of clusters
- if $K>K^*$:
  one of the clusters will break at least one of the "true" clusters;
  the loss function will decrease less
- this leads to the heuristic "kink" (or "elbow") method



> Cluster Analysis / K-means

- ✓ very simple, flexible, efficient
- ✓ fast, O(N) complexity
- ✗ need to input *K*
- ✗ works for well-shaped clusters
- ✗ sensitive to initial conditions
- ✗ sensitive to outliers

- no need to input number of clusters (K),
  but need measure of *dissimilarity* (or proximity) between clusters

- clusters at each level of the hierarchy come from
  merging/dividing clusters at previous level

- **Agglomerative:**
  - start with every point being a 1-point cluster (singleton);
  - at each step the two closest clusters are <u>merged</u> into a single one;
  - stop when one cluster encloses all points.

- **Divisive:**
  - start with one cluster enclosing all points;
  - at each step the clusters are <u>split</u> into two based on dissimilarity;
  - stop when all clusters are singletons, or zero intra-cluster dissimilarity

[from now on, agglomerative clustering]

Agglomerative Clustering Algorithm

- **Input:** set of *N* examples (points)
- **Input:** dissimilarity measure *D* between clusters

1. each point is a 1-point cluster (singleton)

2. compute dissimilarity matrix between all points

3. repeat {

    - merge the two closest clusters into one cluster

    - update dissimilarity matrix

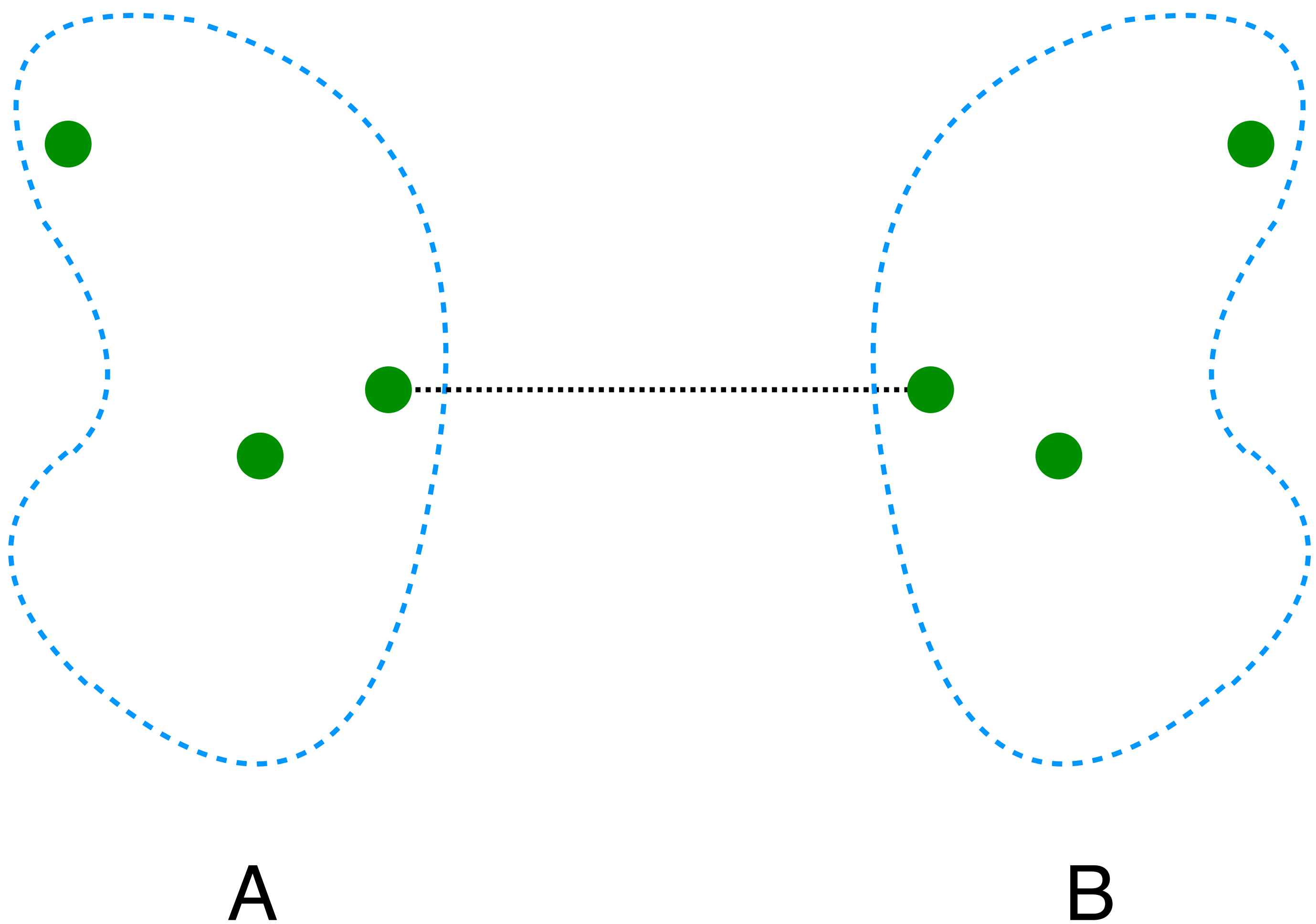
   } until all points in one cluster

Single Link Proximity

[a.k.a. Nearest-neighbor technique]

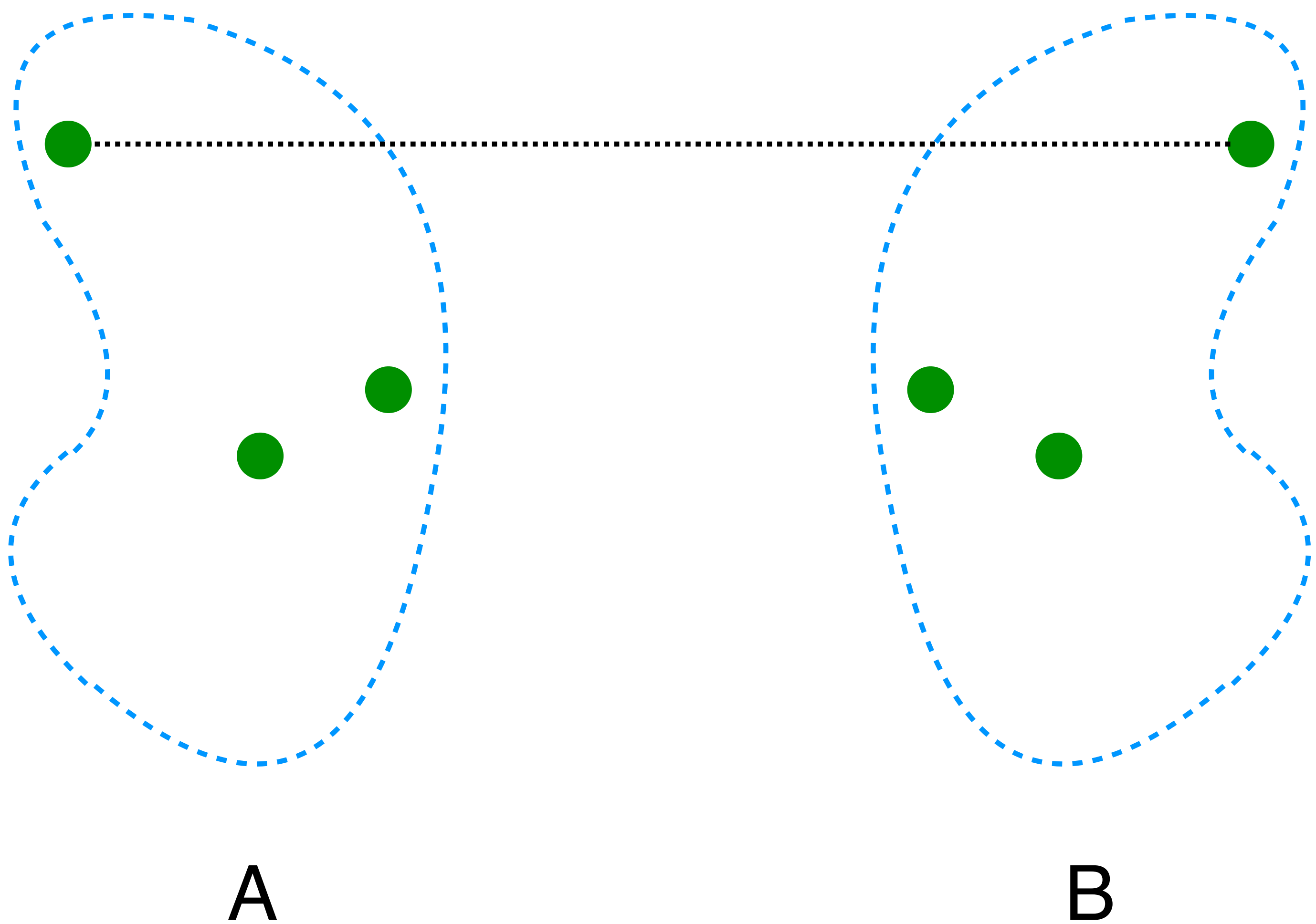
$$D(A,B) = \min_{x\in A, y\in B} d(x,y)$$

e.g. Euclidean distance



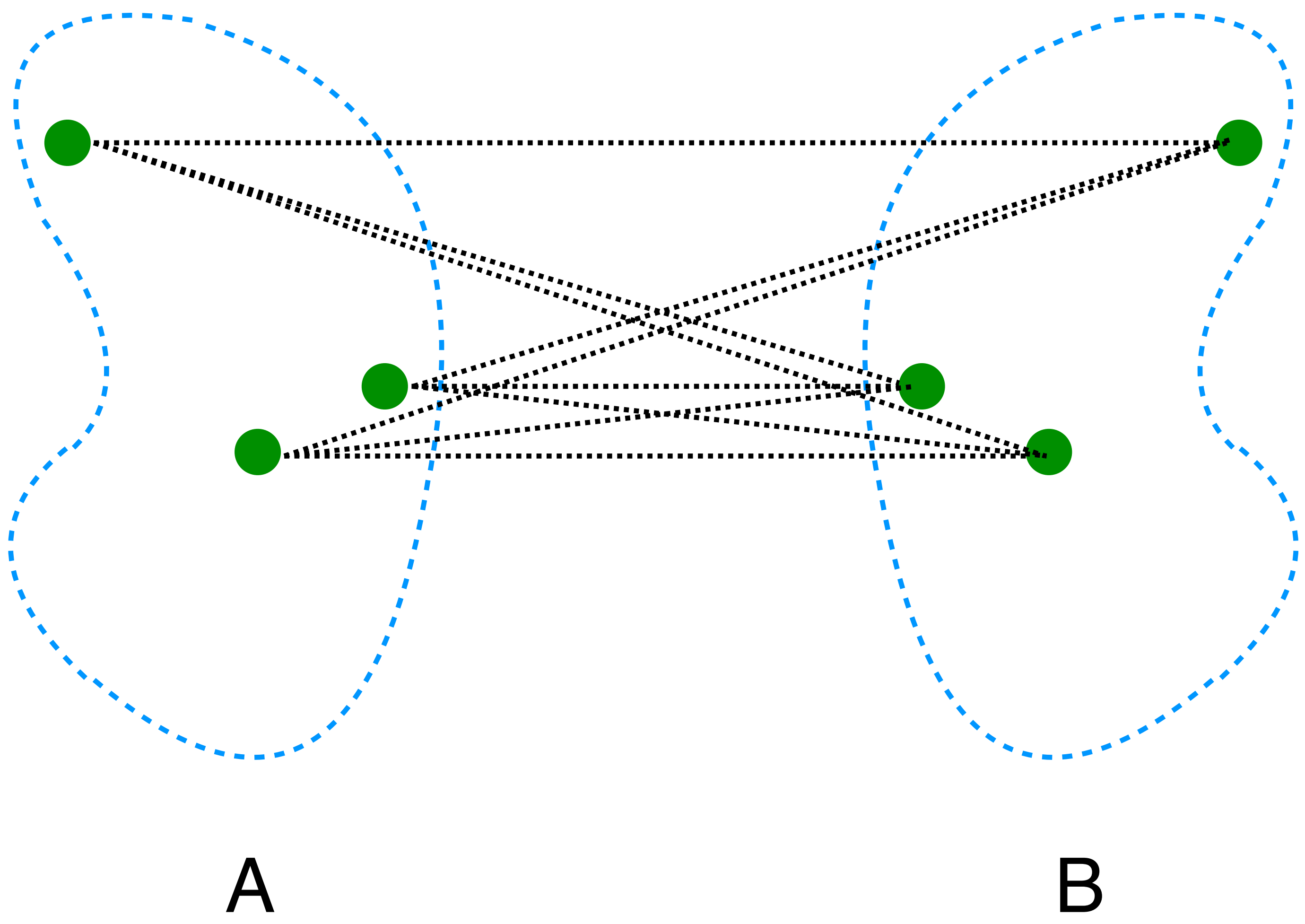
Complete Link Proximity

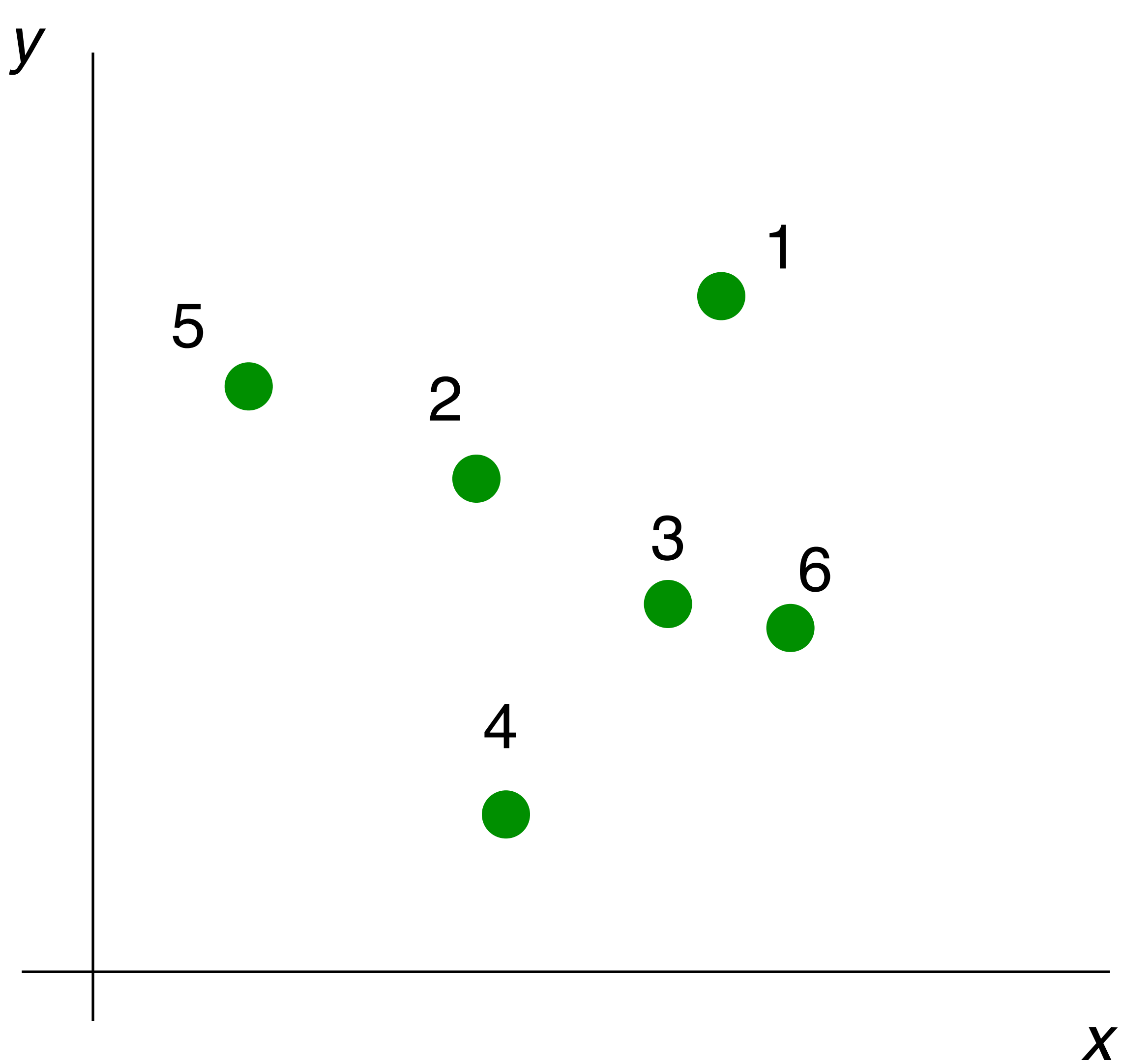
$$D(A,B) = \max_{x \in A, y \in B} d(x,y)$$



Group Average Proximity

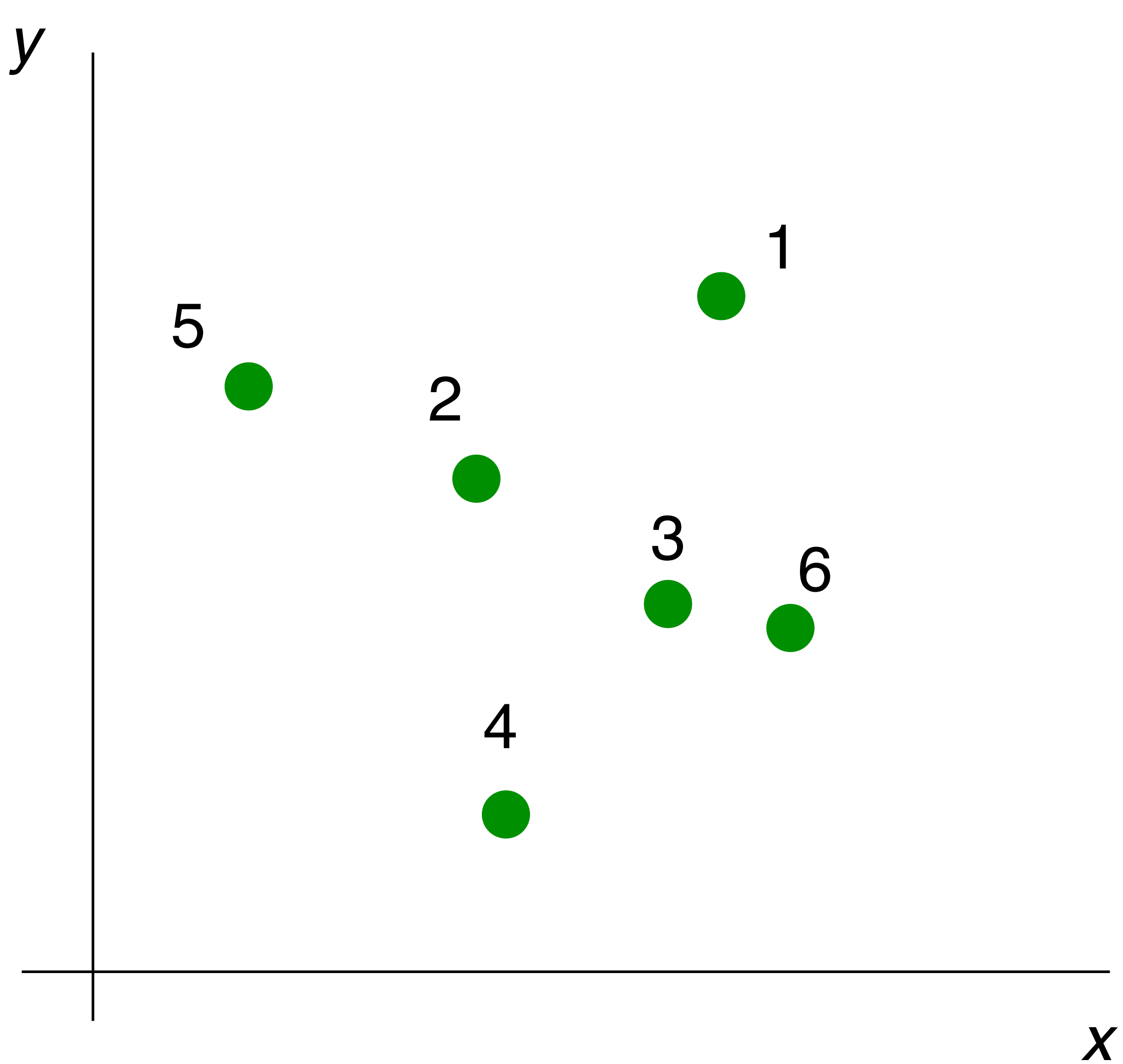
$$D(A,B) = \frac{1}{N_A N_B} \sum_{x \in A} \sum_{y \in B} d(x,y)$$





Coordinates

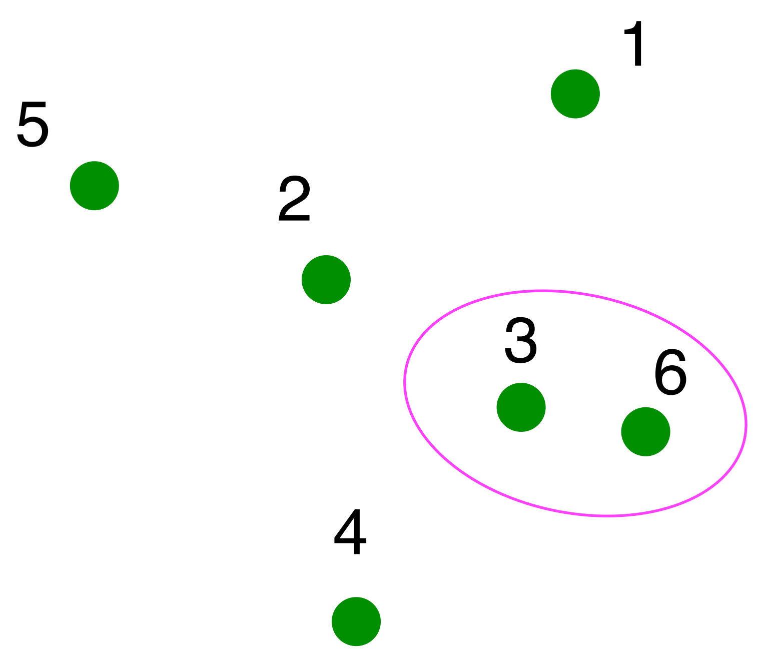
	x	y
1	0.40	0.53
2	0.22	0.38
3	0.35	0.32
4	0.26	0.19
5	0.08	0.41
6	0.45	0.30



Euclidean distances

	1	2	3	4	5	6
1	0	0.23	0.22	0.37	0.39	0.24
2		0	0.14	0.19	0.16	0.24
3			0	0.16	0.27	0.10
4				0	0.22	0.22
5					0	0.37
6						0

Single Link Proximity

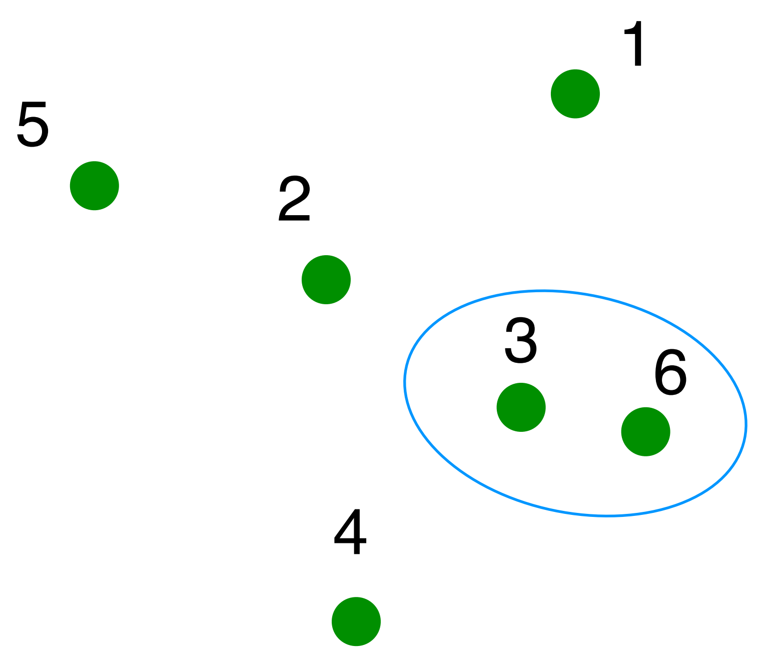


Euclidean distances

	1	2	3	4	5	6
1	0	0.23	0.22	0.37	0.39	0.24
2		0	0.14	0.19	0.16	0.24
3			0	0.16	0.27	0.10
4				0	0.22	0.22
5					0	0.37
6						0

Points 3 and 6: smallest distance.
Merge them into cluster.

Single Link Proximity

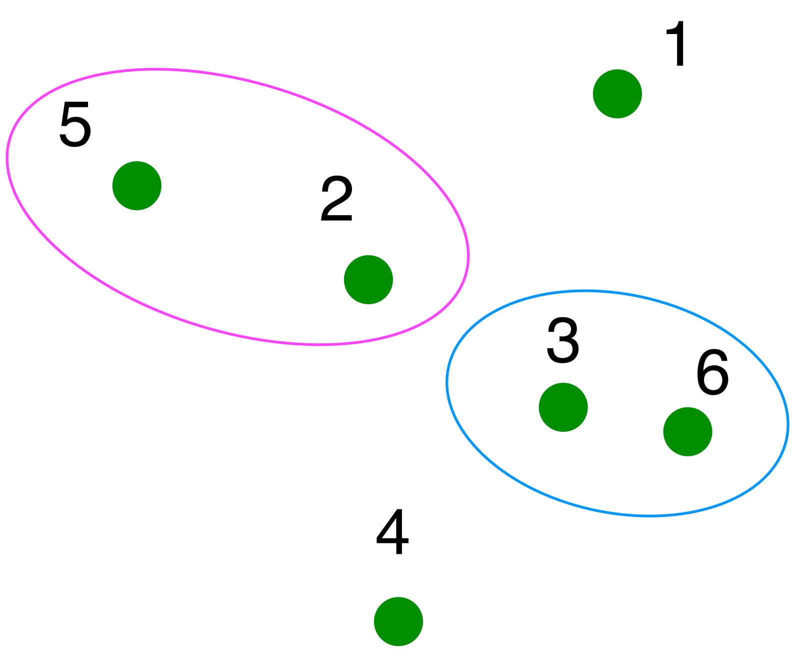


Euclidean distances

	1	2	4	5	3,6
1	0	0.23	0.37	0.39	0.22
2		0	0.19	0.16	0.14
4			0	0.22	0.16
5				0	0.27
3,6					0

Update distances (distance to cluster = **min** distance to its constituents).

Single Link Proximity

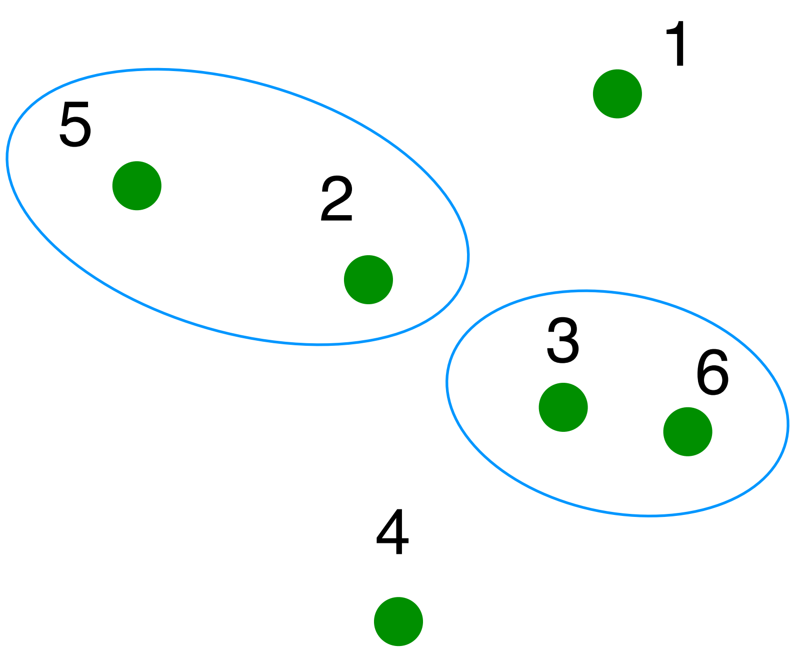


Euclidean distances

	1	2	4	5	3,6
1	0	0.23	0.37	0.39	0.22
2		0	0.19	0.16	0.14
4			0	0.22	0.16
5				0	0.27
3,6					0

Iterate until you include all points into one cluster

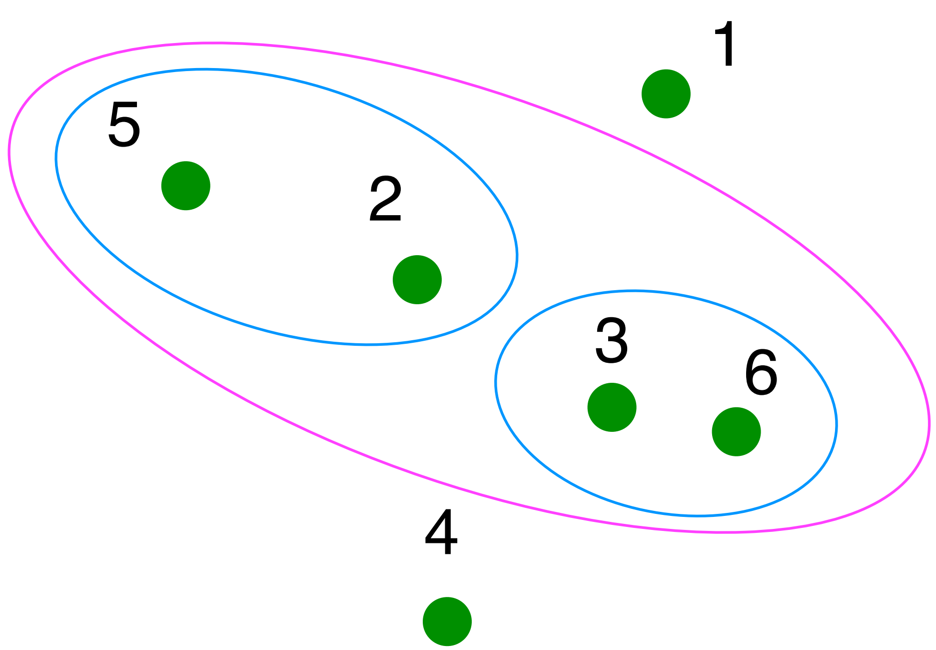
Single Link Proximity



Euclidean distances

	1	4	2,5	3,6
1	0	0.37	0.23	0.22
4		0	0.19	0.15
2,5			0	0.14
3,6				0

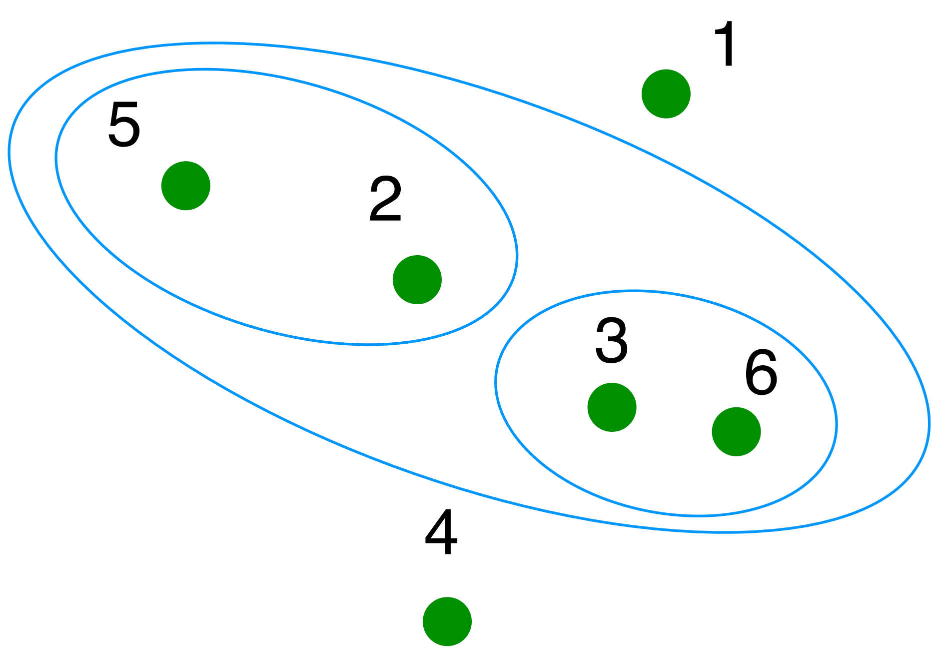
Single Link Proximity



Euclidean distances

	1	4	2,5	3,6
1	0	0.37	0.23	0.22
4		0	0.19	0.15
2,5			0	0.14
3,6				0

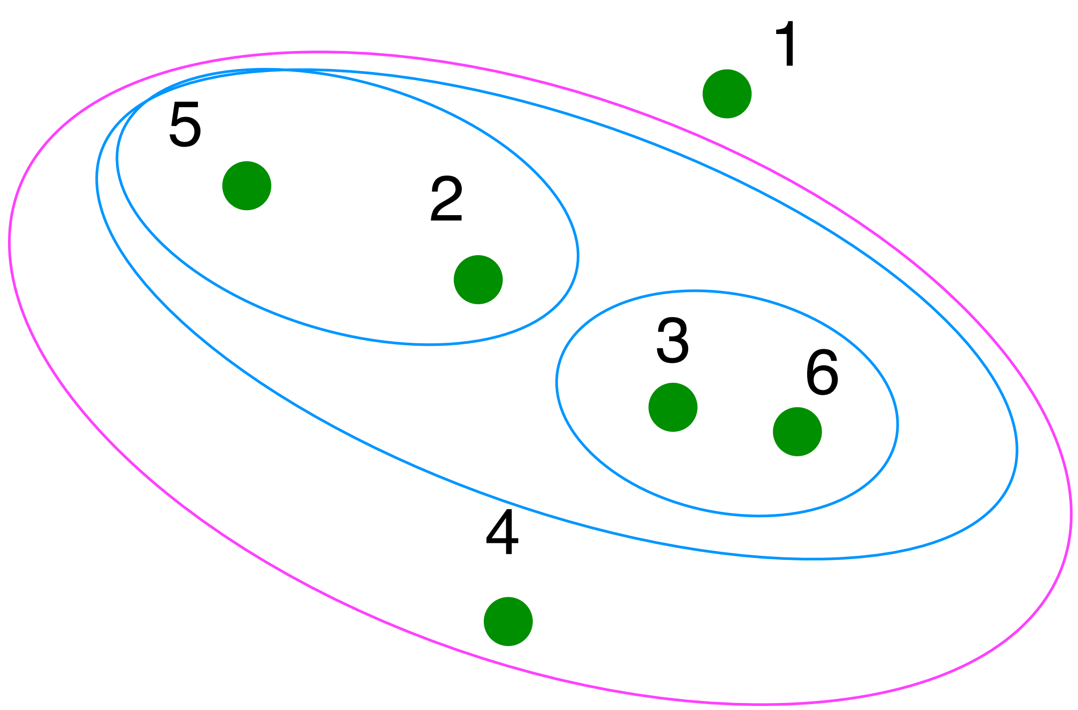
Single Link Proximity



Euclidean distances

	1	4	2,5,3,6
1	0	0.37	0.22
4		0	0.15
2,5,3,6			0

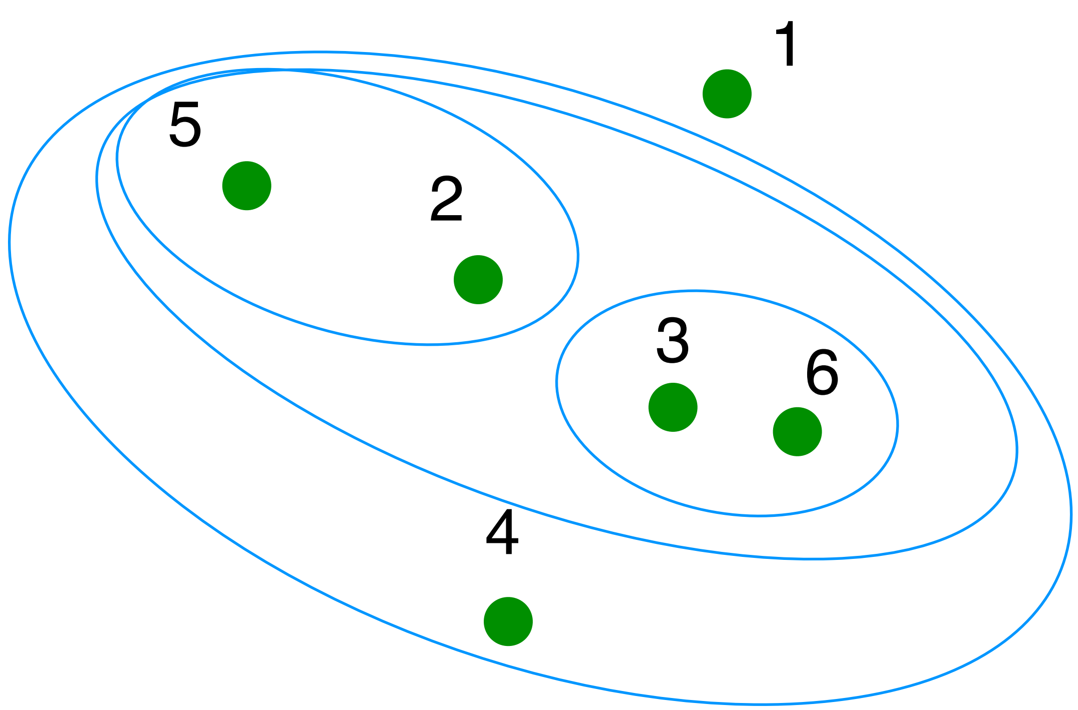
Single Link Proximity



Euclidean distances

	1	4	2,5,3,6
1	0	0.37	0.22
4		0	0.15
2,5,3,6			0

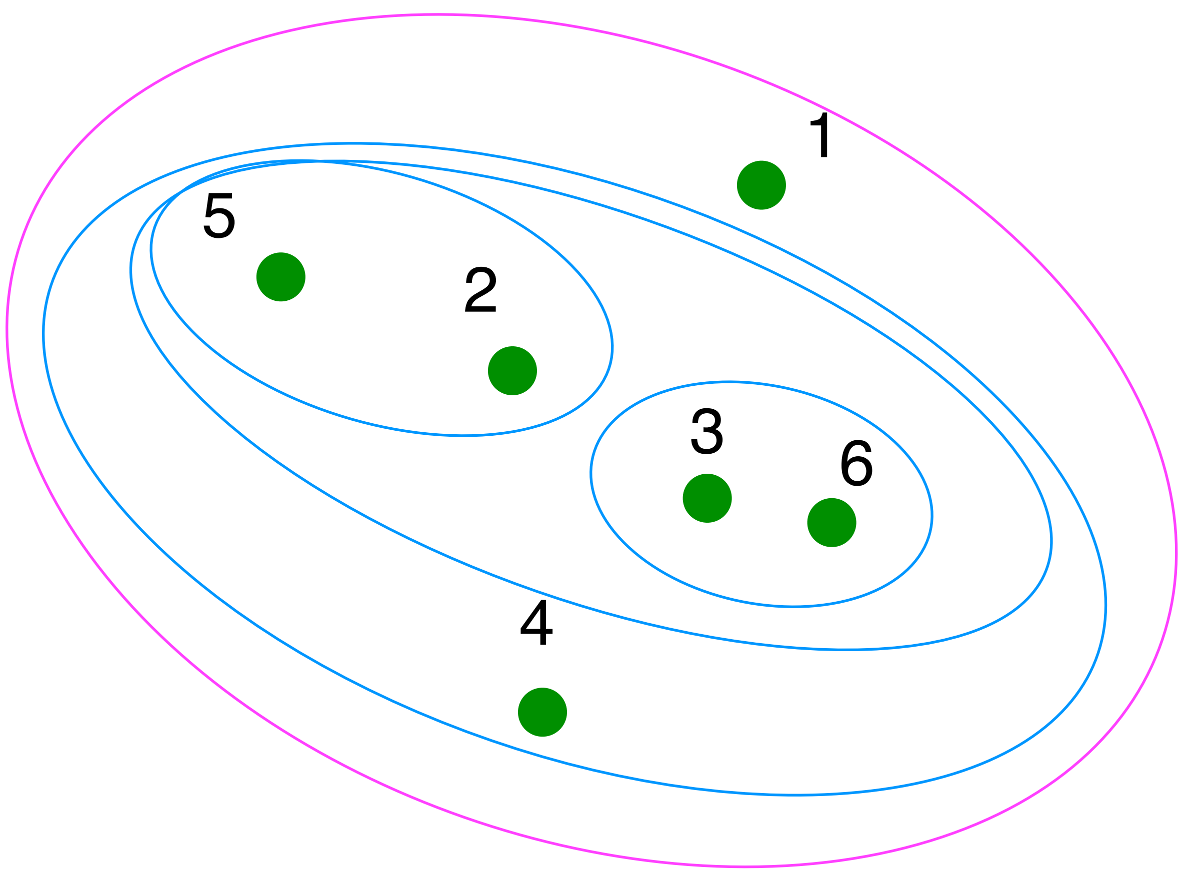
Single Link Proximity



Euclidean distances

	1	4,2,5,3,6
1	0	0.22
4,2,5,3,6		0

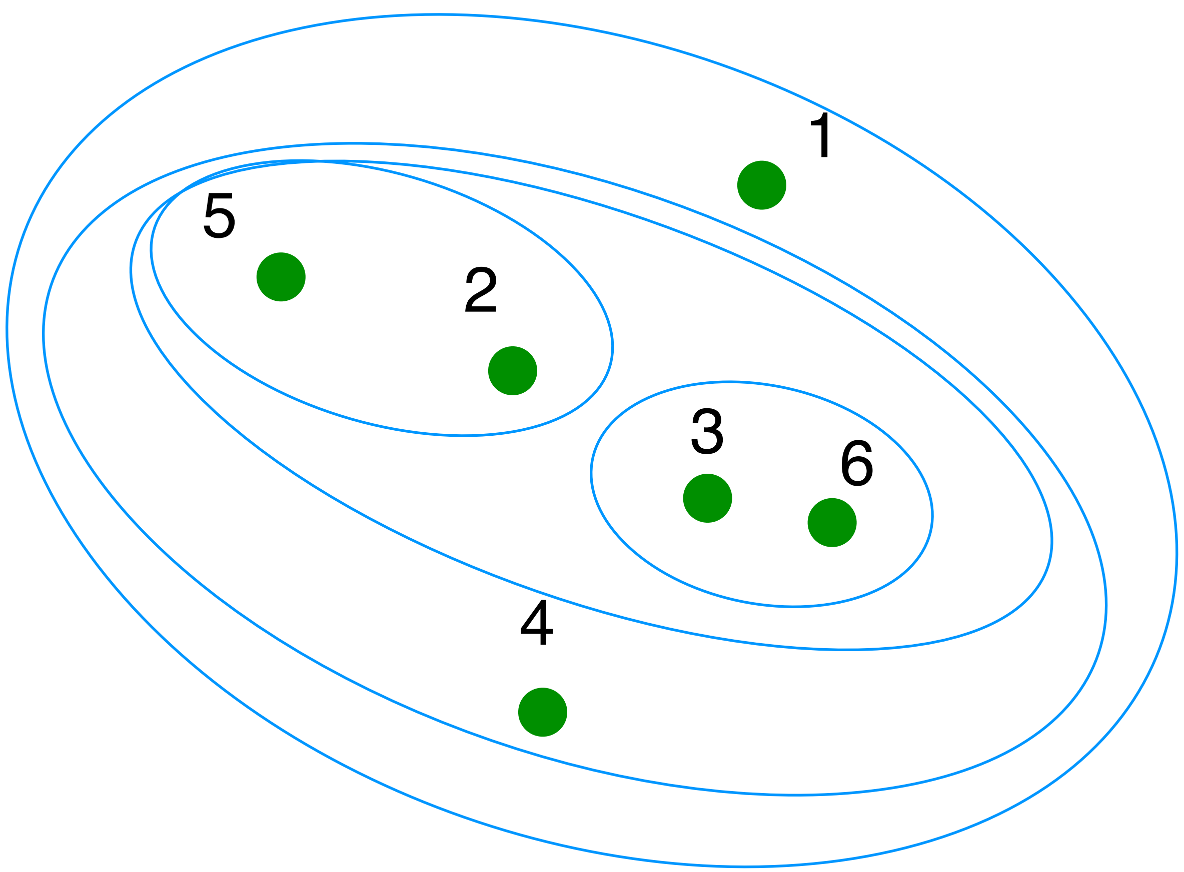
Single Link Proximity



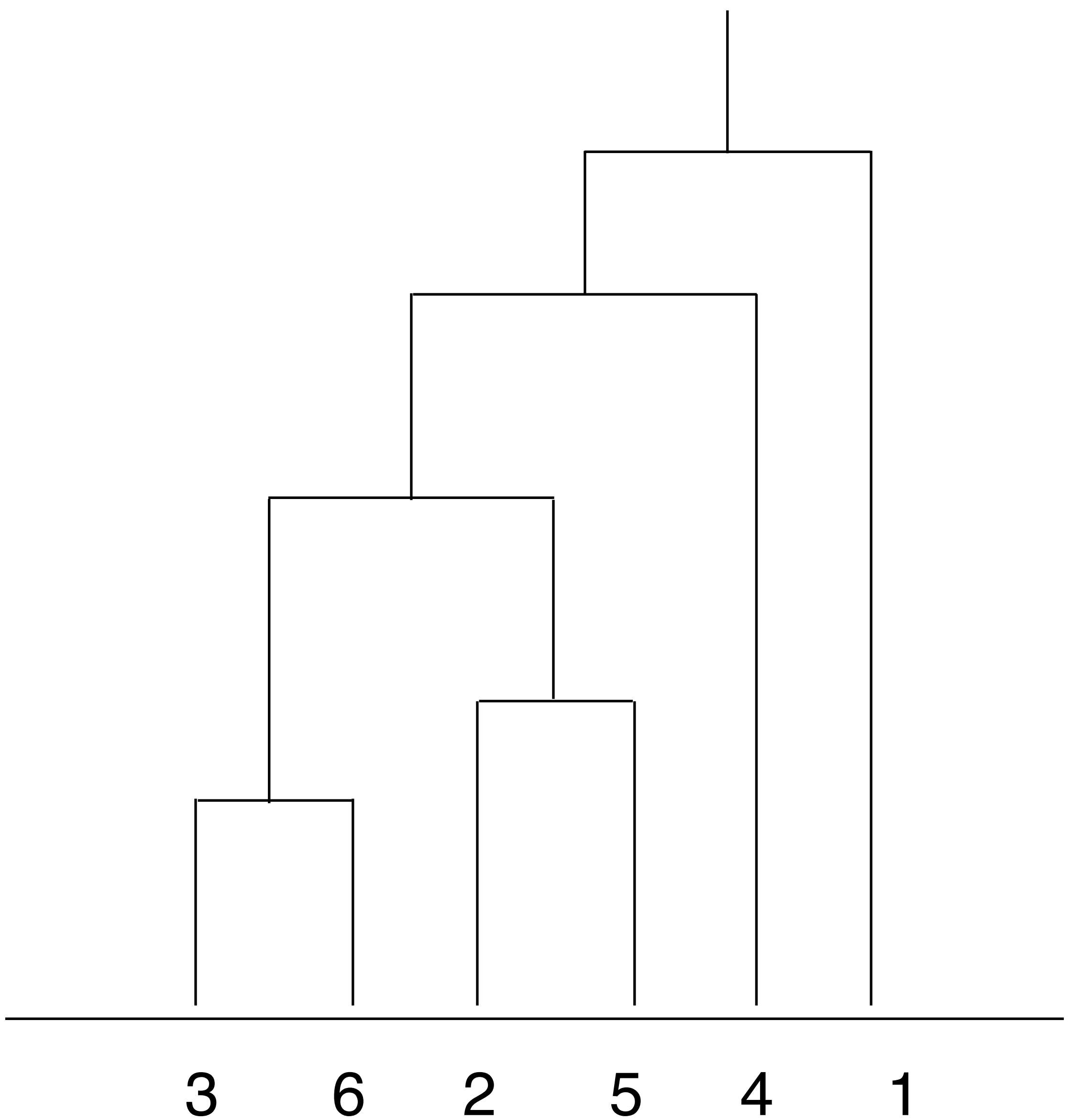
Euclidean distances

	1	4,2,5,3,6
1	0	0.22
4,2,5,3,6		0

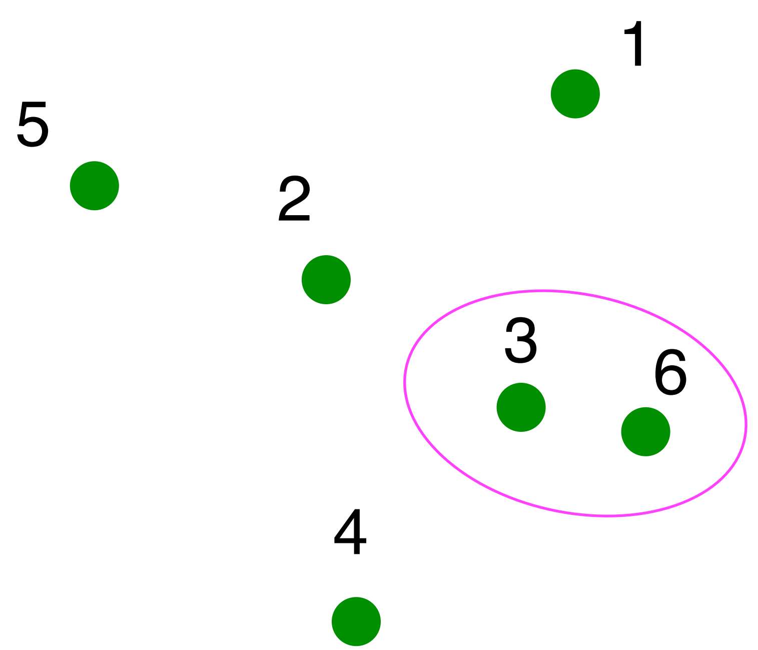
Single Link Proximity



Dendrogram



Complete Link Proximity

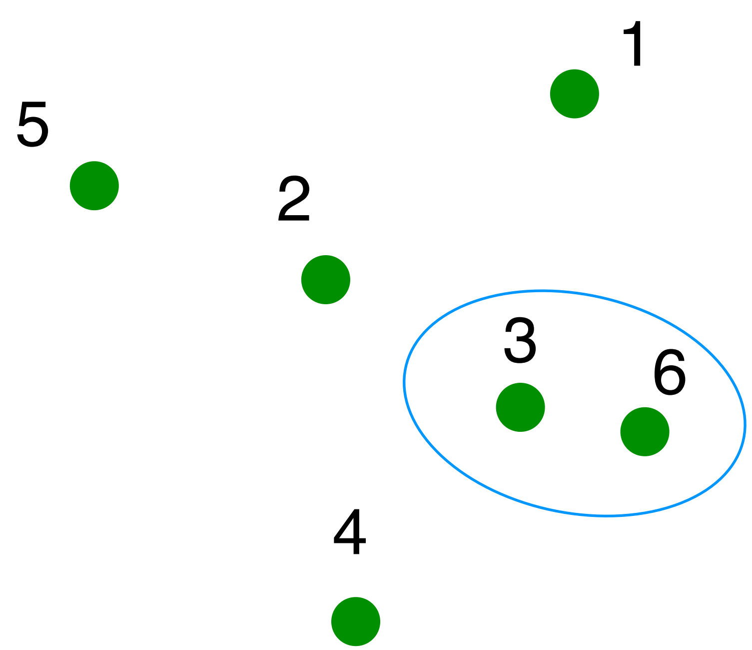


Euclidean distances

	1	2	3	4	5	6
1	0	0.23	0.22	0.37	0.39	0.24
2		0	0.14	0.19	0.16	0.24
3			0	0.16	0.27	0.10
4				0	0.22	0.22
5					0	0.37
6						0

Points 3 and 6: smallest distance.
Merge them into cluster.

Complete Link Proximity

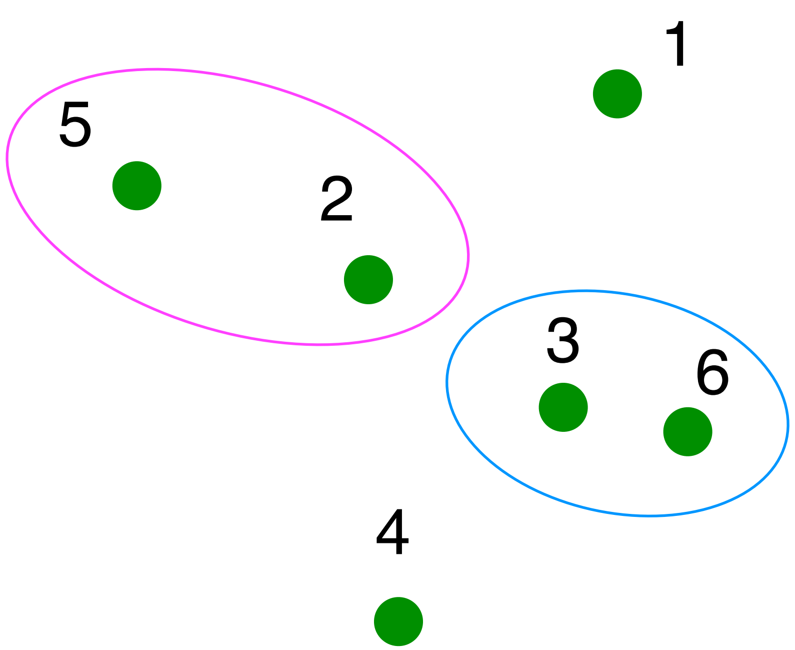


Euclidean distances

	1	2	4	5	3,6
1	0	0.23	0.37	0.39	0.24
2		0	0.19	0.16	0.24
4			0	0.22	0.22
5				0	0.37
3,6					0

Update distances (distance to cluster = **max** distance to its constituents).

Complete Link Proximity

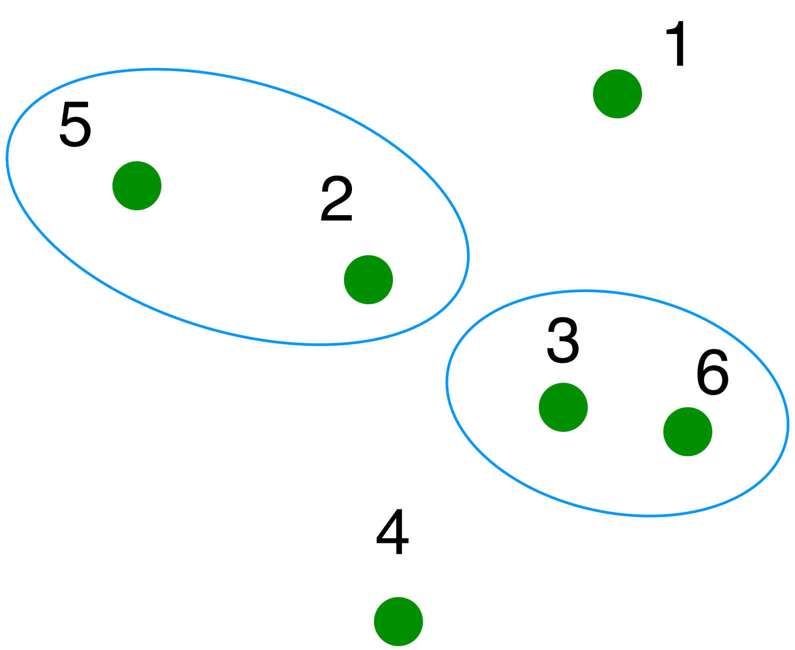


Euclidean distances

	1	2	4	5	3,6
1	0	0.23	0.37	0.39	0.24
2		0	0.19	0.16	0.24
4			0	0.22	0.22
5				0	0.37
3,6					0

Iterate until you include all points into one cluster

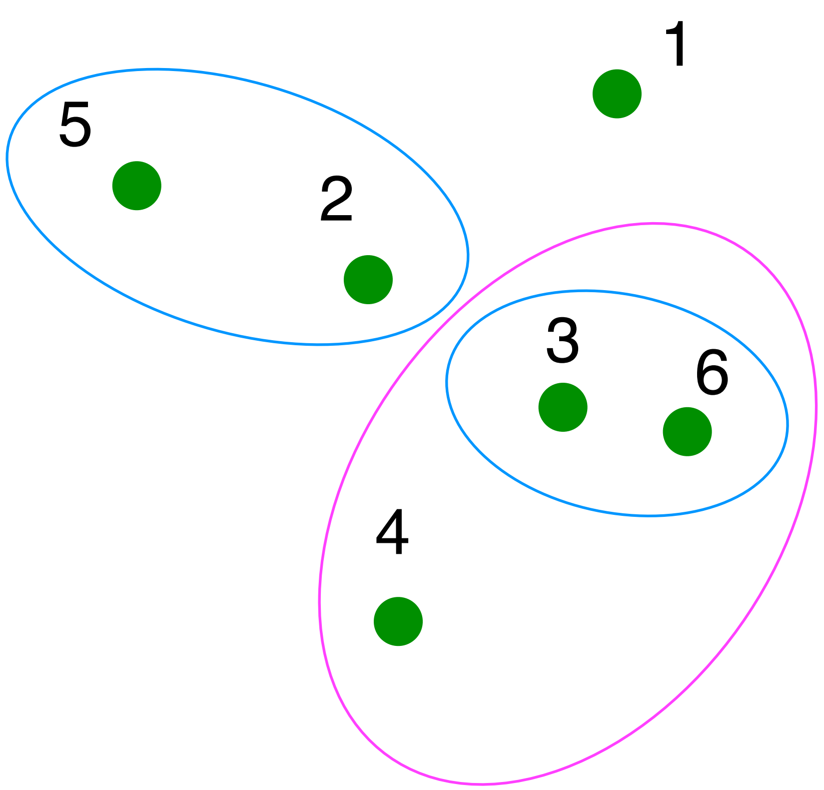
Complete Link Proximity



Euclidean distances

	1	4	2,5	3,6
1	0	0.37	0.39	0.24
4		0	0.22	0.22
2,5			0	0.37
3,6				0

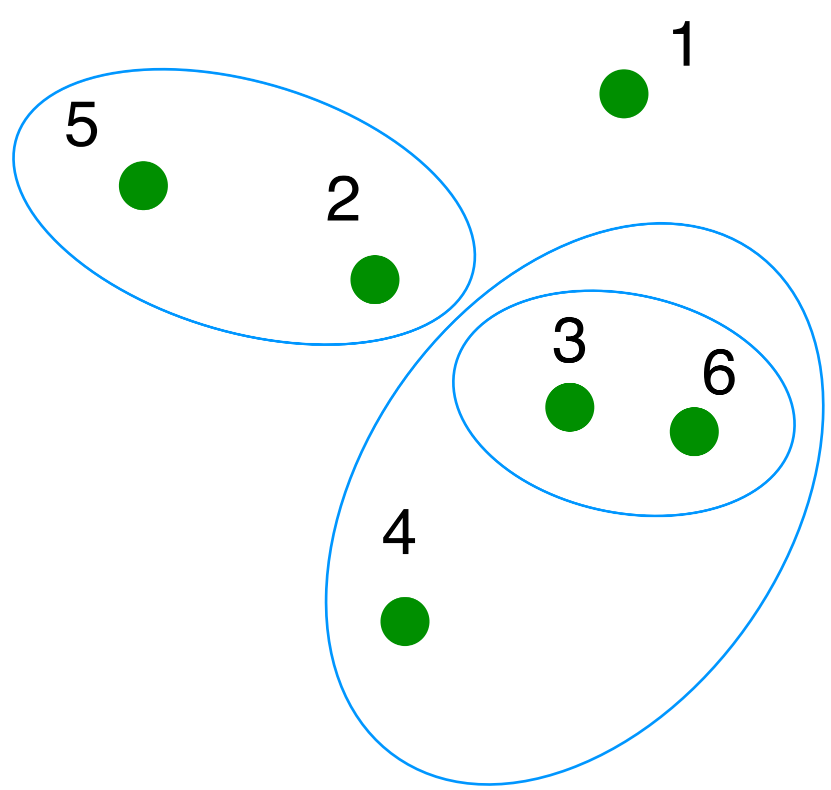
Complete Link Proximity



Euclidean distances

	1	4	2,5	3,6
1	0	0.37	0.39	0.24
4		0	0.22	0.22
2,5			0	0.37
3,6				0

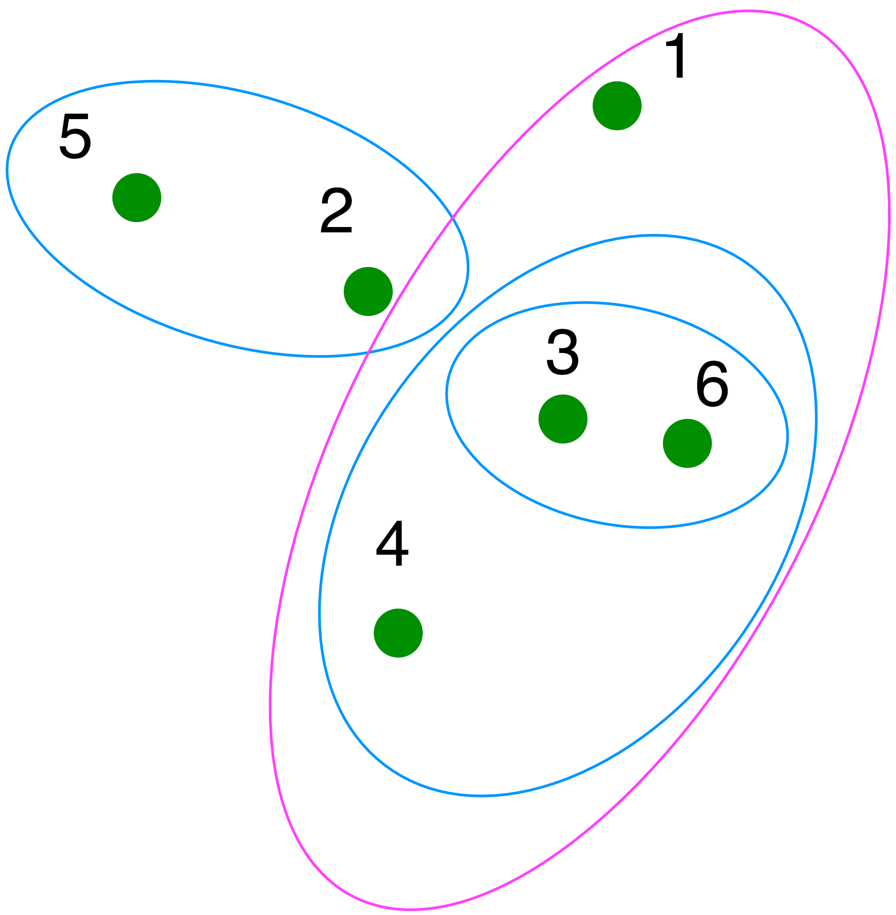
Complete Link Proximity



Euclidean distances

	1	2,5	4,3,6
1	0	0.39	0.37
2,5		0	0.37
4,3,6			0

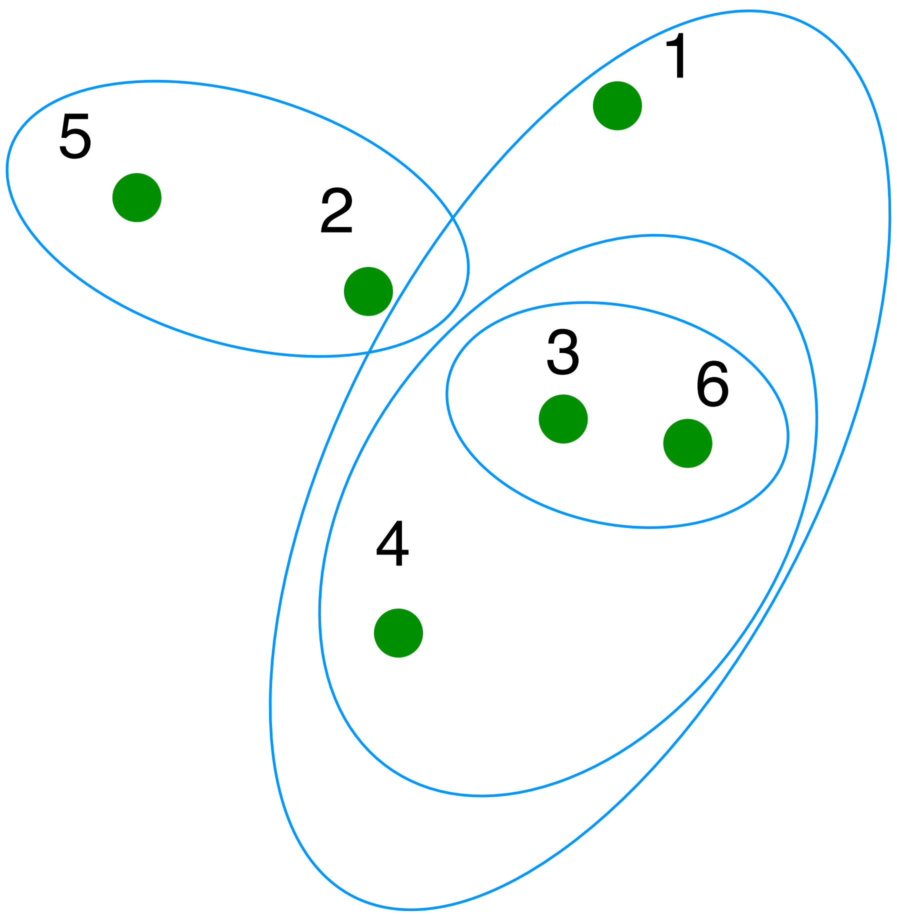
Complete Link Proximity



Euclidean distances

	1	2,5	4,3,6
1	0	0.39	0.37
2,5		0	0.37
4,3,6			0

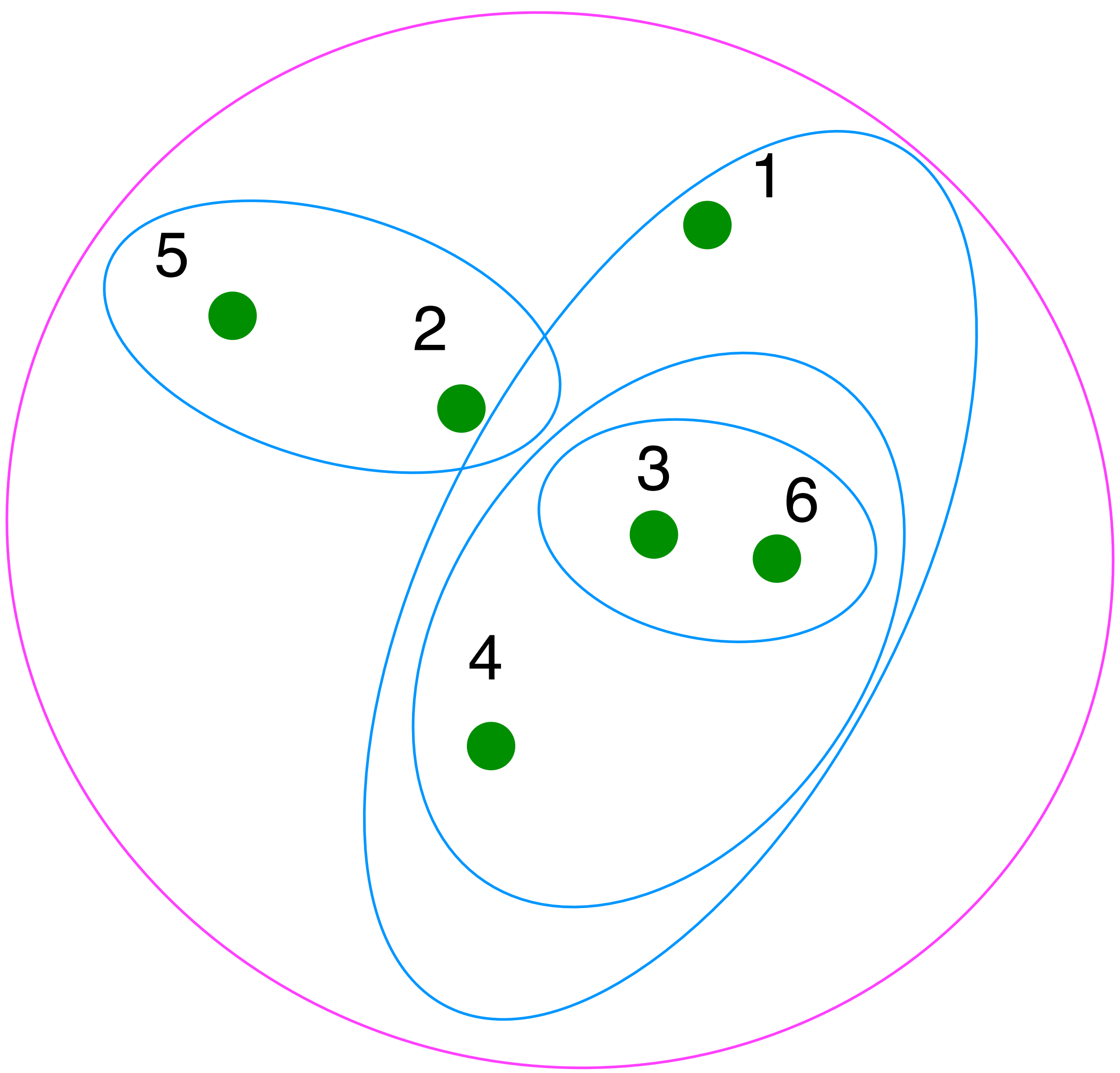
Complete Link Proximity



Euclidean distances

	1,4,3,6	2,5
1,4,3,6	0	0.39
2,5		0

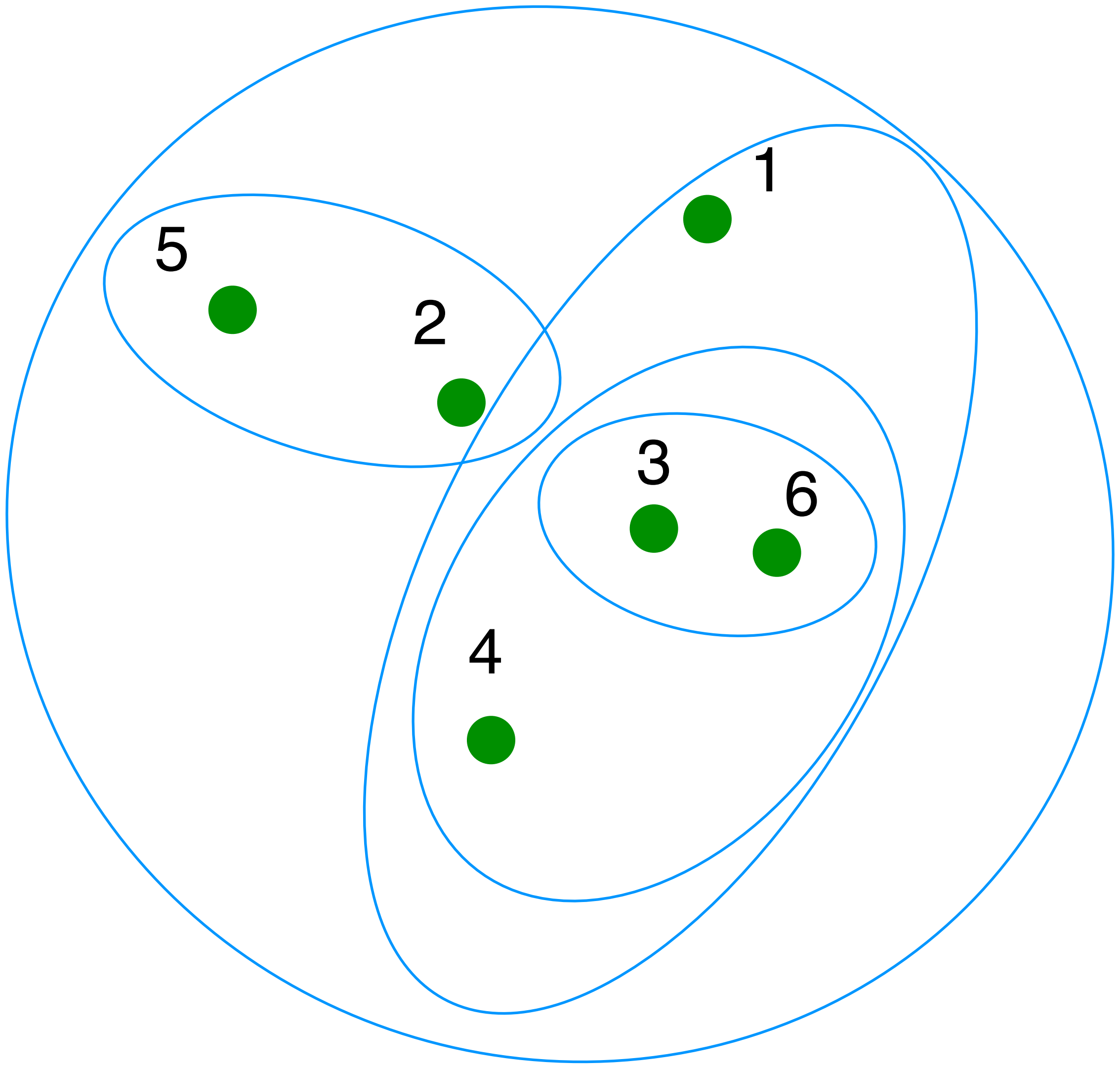
Complete Link Proximity



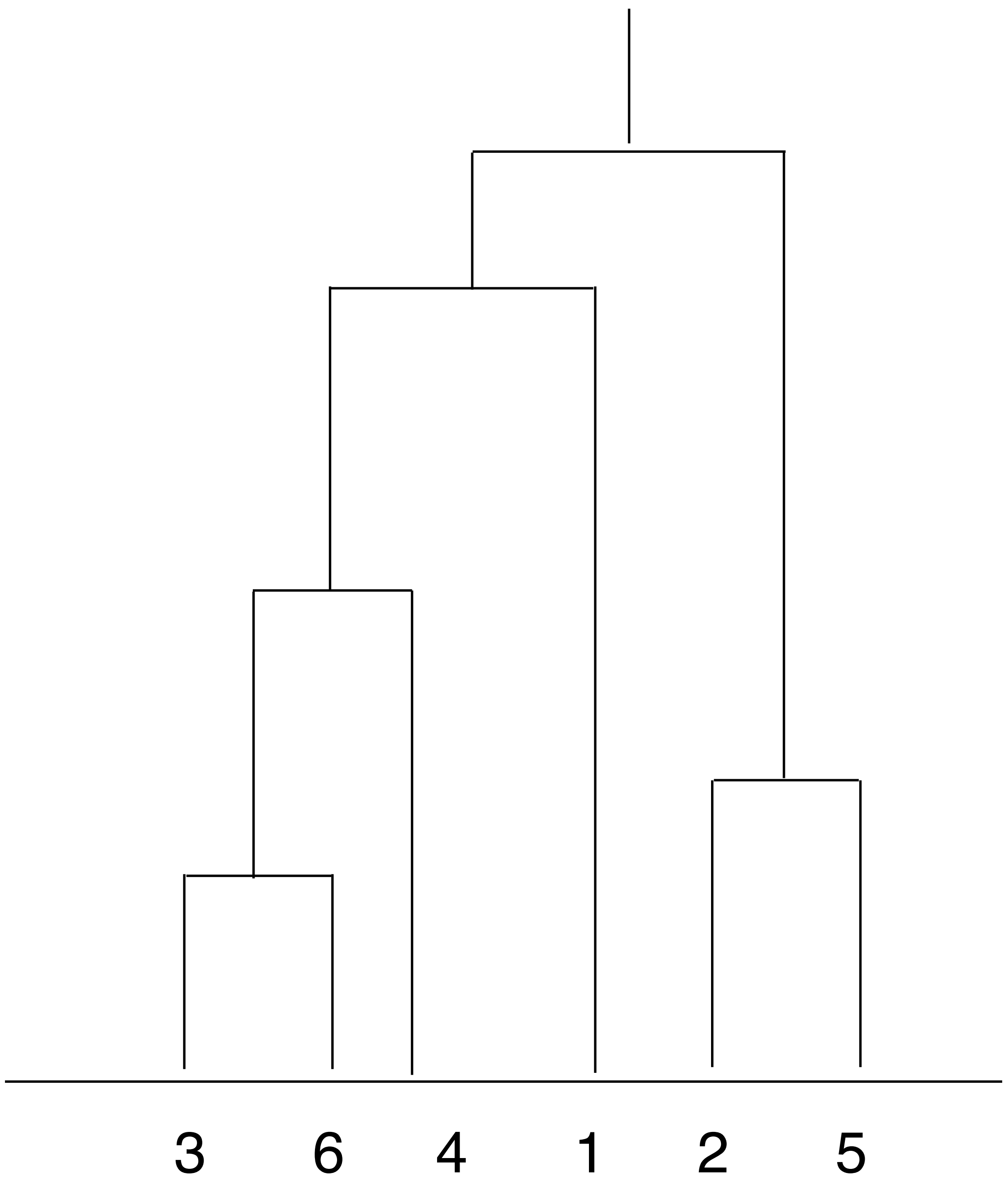
Euclidean distances

	1,4,3,6	2,5
1,4,3,6	0	0.39
2,5		0

Complete Link Proximity

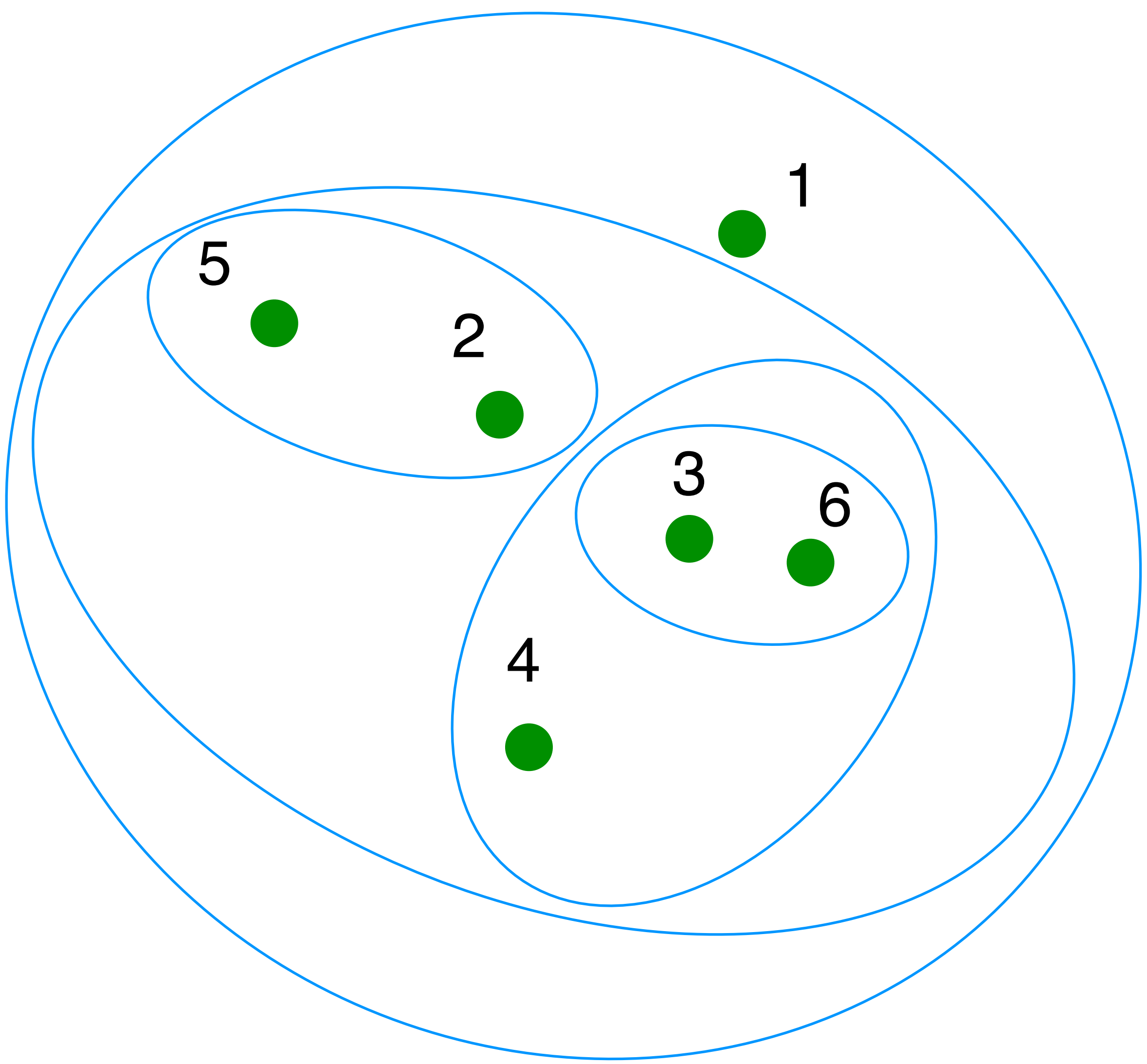


Dendrogram

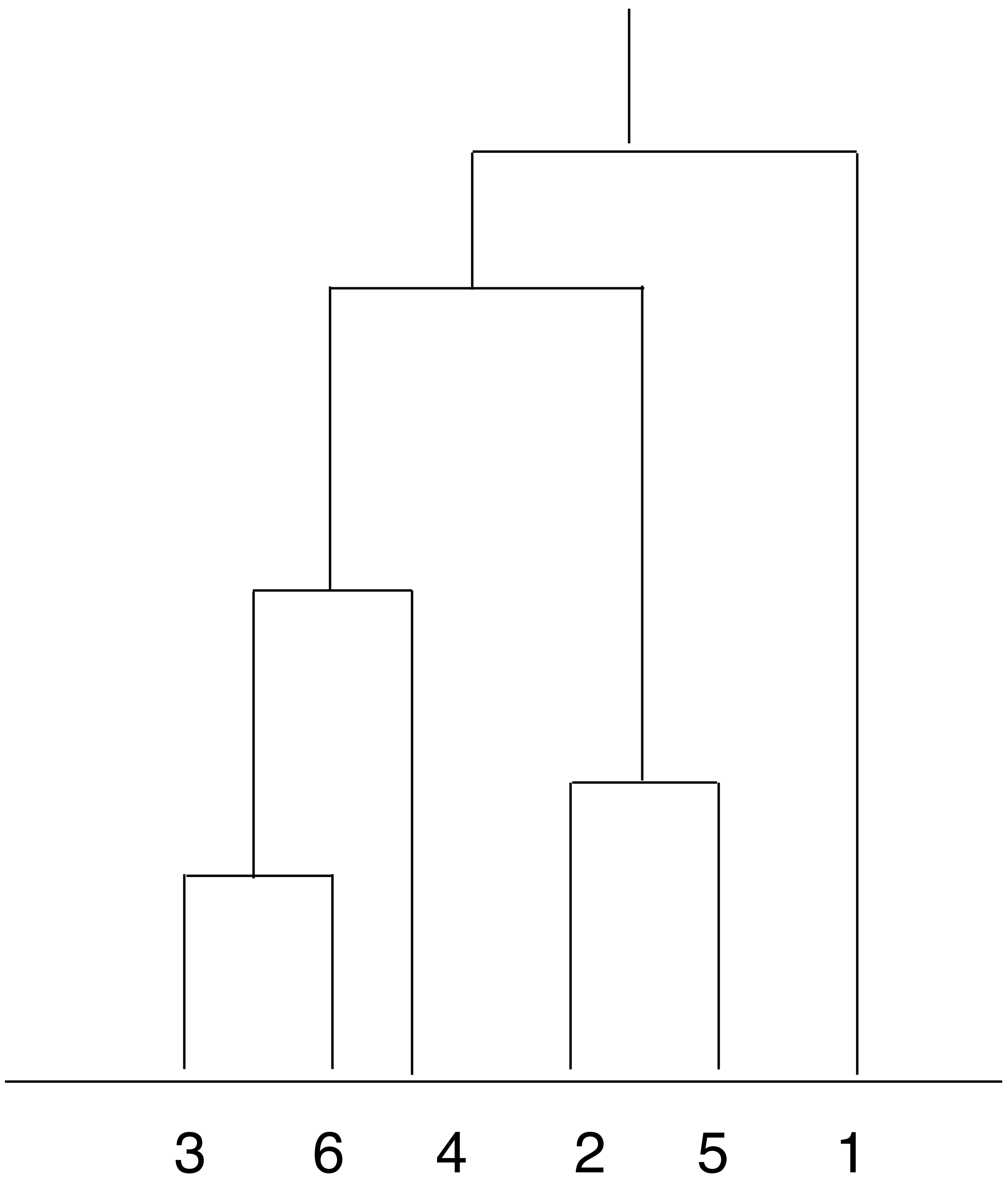


Exercise:

Group Average Proximity



Dendrogram



Summary

diameter of a cluster A: $D_A = \max_{x \in A, y \in A} d(x,y)$

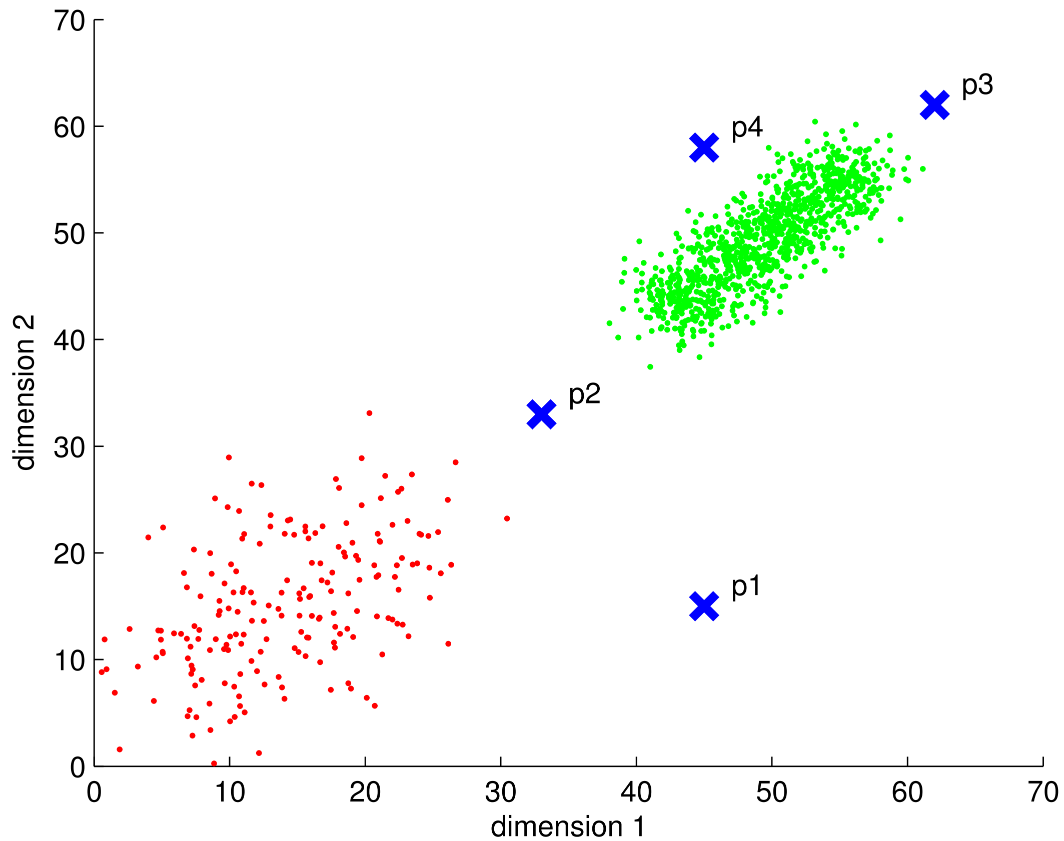
is the largest inter-cluster dissimilarity

- Single link: tend to produce close clusters (with large diameters)
- Complete link: tend to produce compact clusters (with small diameters)
- Group average: compromise situation

- ✓ easy to implement
- ✓ no need to input number of clusters
- ✘ $O(N^2 \log(N))$ complexity
- ✘ no minimization objective
- ✘ unable to undo previous merges

2
ANOMALY DETECTION

What is an Anomaly?



Anomalies and **Outliers** are basically the same thing: objects that are different from most other objects

"An outlier is an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism" [D. Hawkins - 1990]

Approaches to Anomaly Detection

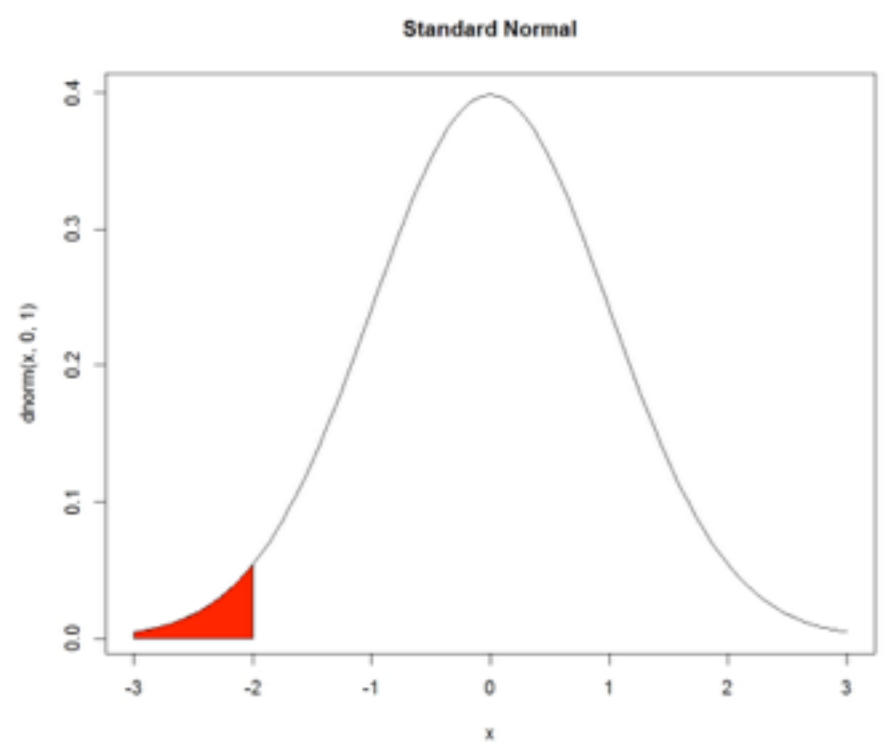
- Model-Based:
  - build model of the data
  - anomalies are objects not fitting the model well

- Distance (or Proximity)-Based:
  - compute distances between any pair of points
  - anomalies are objects distant from most of the others

- Density-Based:
  - estimate local density of points
  - anomalies are points lying in low-density regions

Model-based approach

idea: outliers occur at the tails of the prob. distribution



- model a probability distribution *P(x)* from data
- compute the probability for points under *P(x)*
- if $P(x_i) < \varepsilon$, then x_i is an outlier

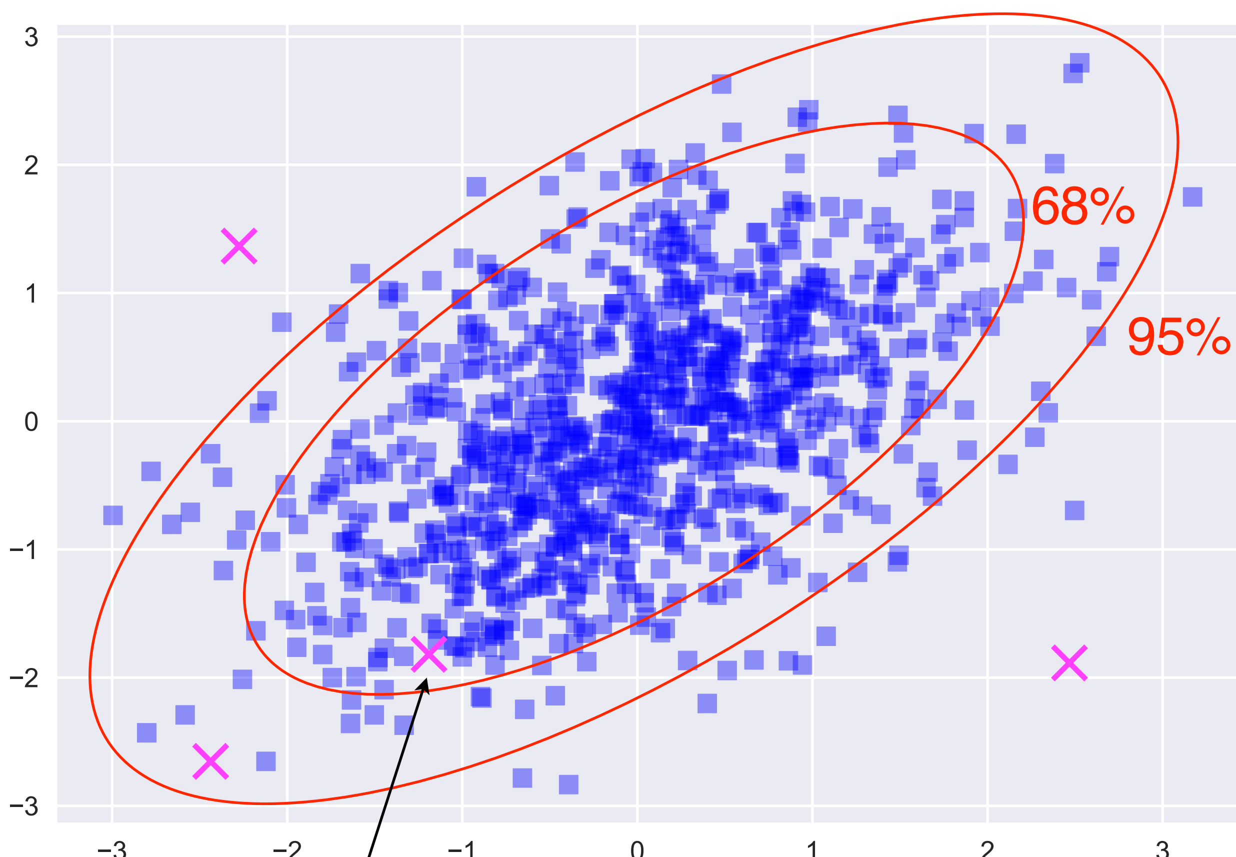
✓ very simple
✓ solid statistical foundation
✘ need to infer the model *P(x)*
✘ poor in high-D

Multi-variate gaussian model

$$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{D}{2}}\det(\Sigma)^{\frac{1}{2}}}\exp\left[-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right]$$

$$\mu = \begin{bmatrix}0\\0\end{bmatrix}$$

$$\Sigma = \begin{bmatrix}1 & 0.5\\0.5 & 1\end{bmatrix}$$



$$\{x^{(1)},\ldots,x^{(N)}\}, \quad x\in\mathbb{R}^D$$

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x^{(i)}$$

$$\Sigma = \frac{1}{N}\sum_{i=1}^{N}(x^{(i)}-\mu)(x^{(i)}-\mu)^T$$

$x^{(k)}$ is anomalous if $\boxed{p(x^{(k)};\mu,\Sigma) < \epsilon}$

hard to detect anomalies
close to normal points

Distance-based approach

idea: outliers are far apart from their neihgbors

- define a distance (proximity) measure $D(x,y)$
- compute distance $D(x_i,x_j)$ for any pair of points
- compute *outlier score:* $S(x_i) = f(\, D(x_i,x_j)\,)$, for any x_i,x_j
- if $S(x_i) > \mathit{threshold}$, then x_i is an outlier
- or top n points with largest $S(x_i)$ are outliers

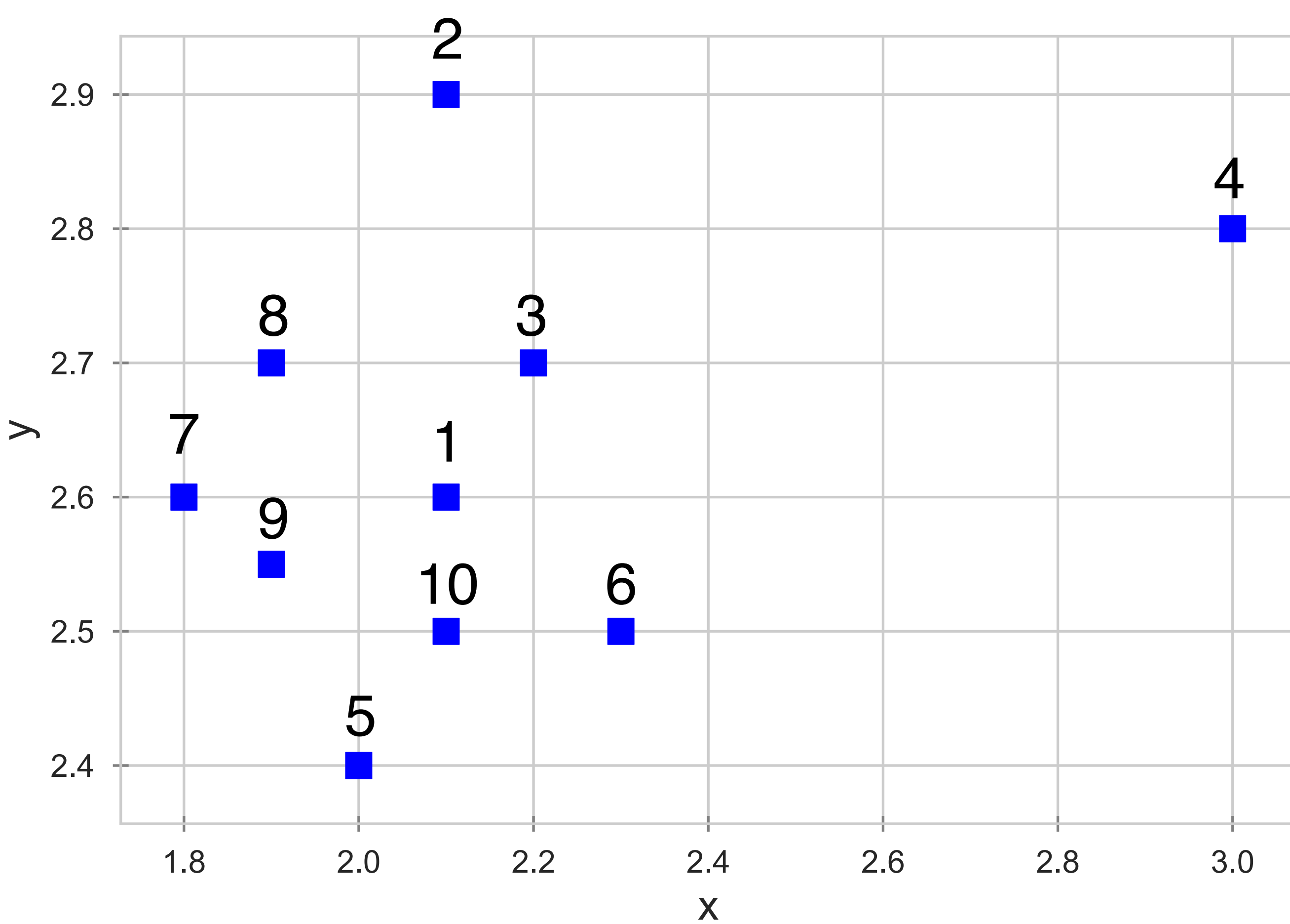
✓ simple and intuitive
✓ more general than model-based
✗ complexity typically ~ $O(n^2)$
✗ sensitive to parameter choices
✗ poor for widely different densities

Nested-Loop K-NN

Outlier scoring based on distances of **nearest neighbors** (NN), e.g.

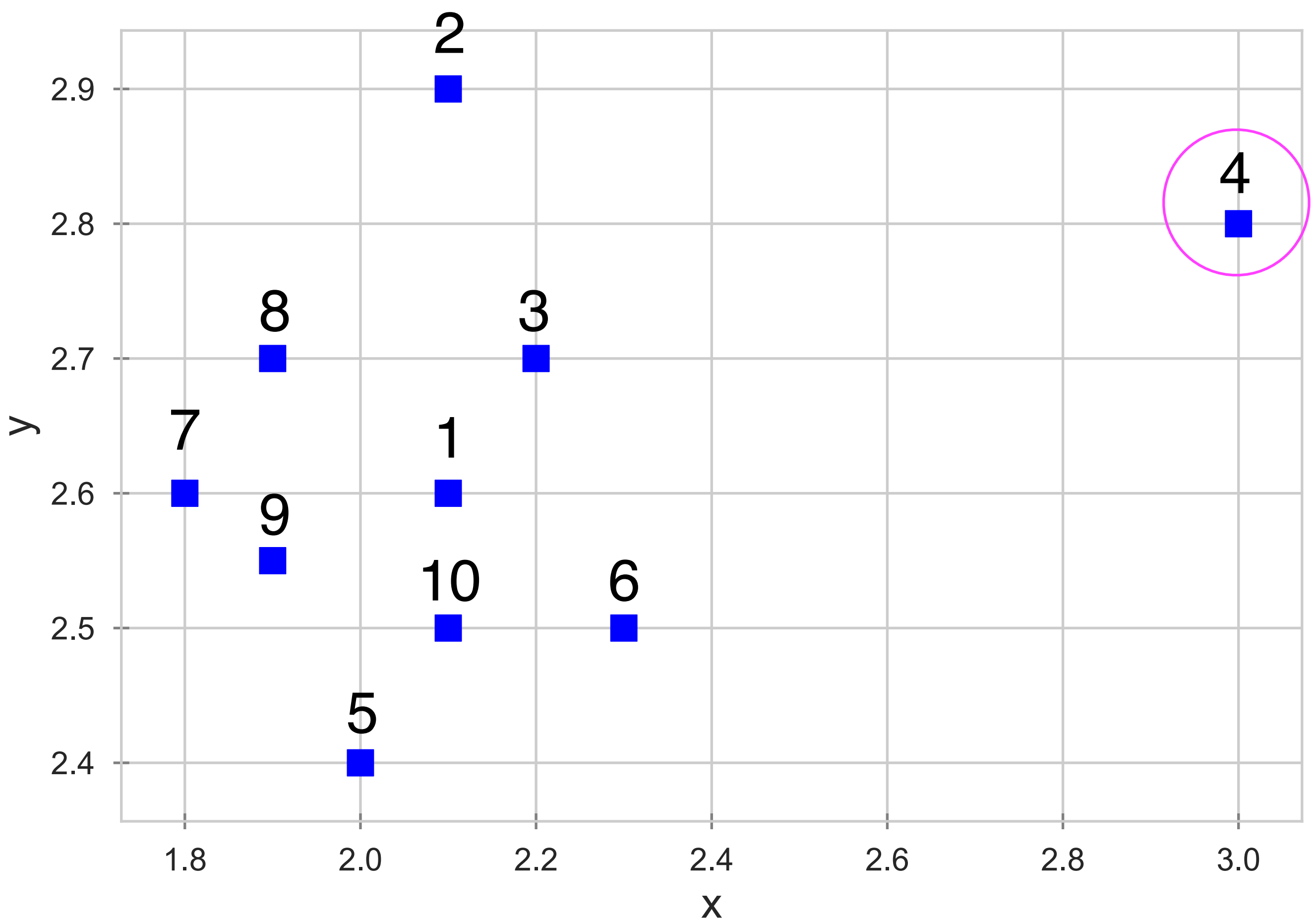
- distance to the k^{th}-NN
- avg distance of the k-NN
- number of NN within distance r

Resolution adjusted with parameter k *(or r)*



Coordinates

	x	y
1	2.1	2.6
2	2.1	2.9
3	2.2	2.7
4	3.0	2.8
5	2.0	2.4
6	2.3	2.5
7	1.8	2.6
8	1.9	2.7
9	1.9	2.55
10	2.1	2.5

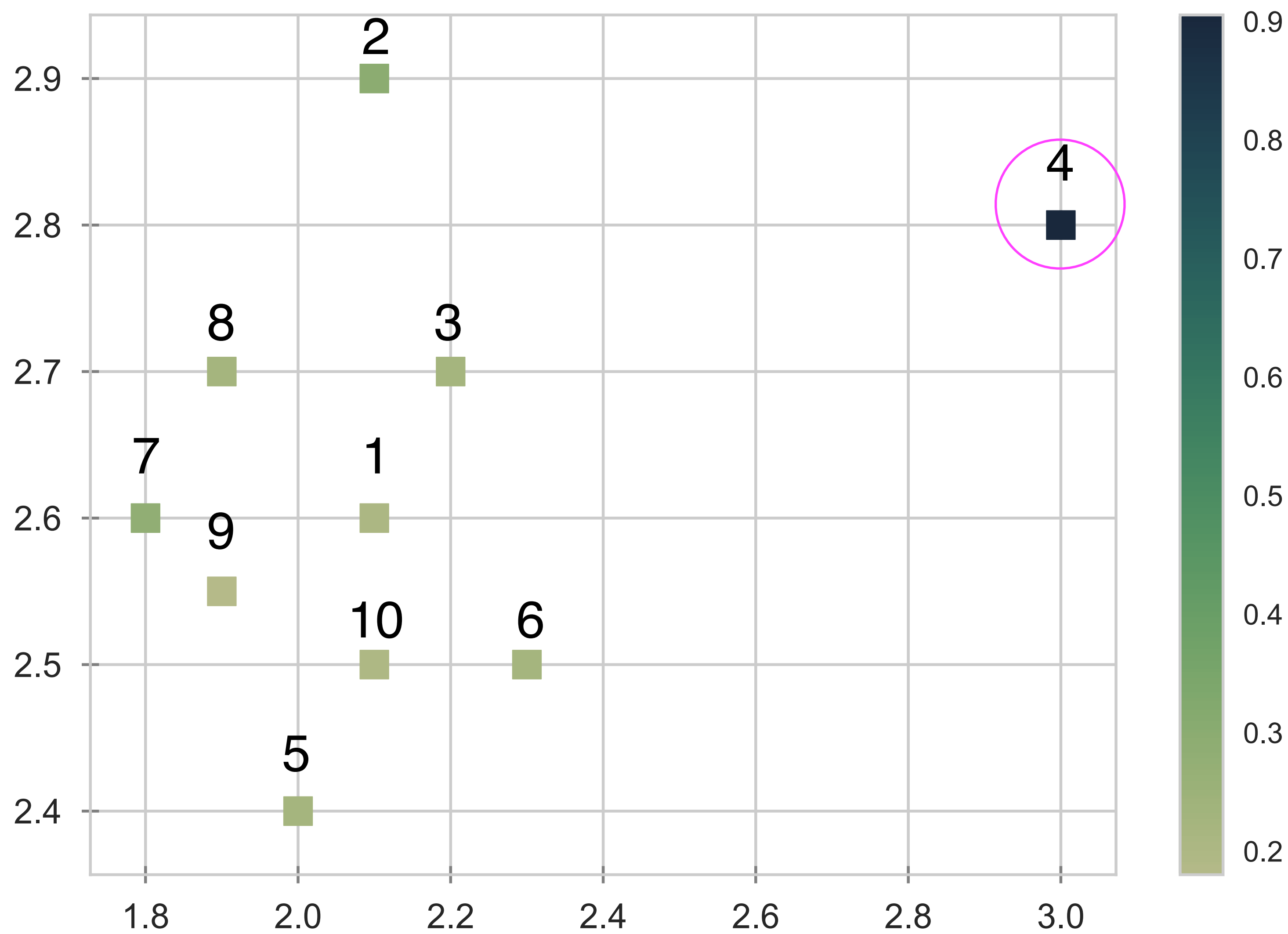


distances to *k*-NN

k=3	1-NN	2-NN	3-NN
1	0.10	0.14	0.21
2	0.22	0.28	0.30
3	0.14	0.22	0.22
4	0.76	0.81	0.91
5	0.14	0.18	0.22
6	0.20	0.22	0.22
7	0.11	0.14	0.28
8	0.14	0.15	0.22
9	0.11	0.15	0.18
10	0.10	0.14	0.20

> Anomaly Detection / Distance-based

k=3

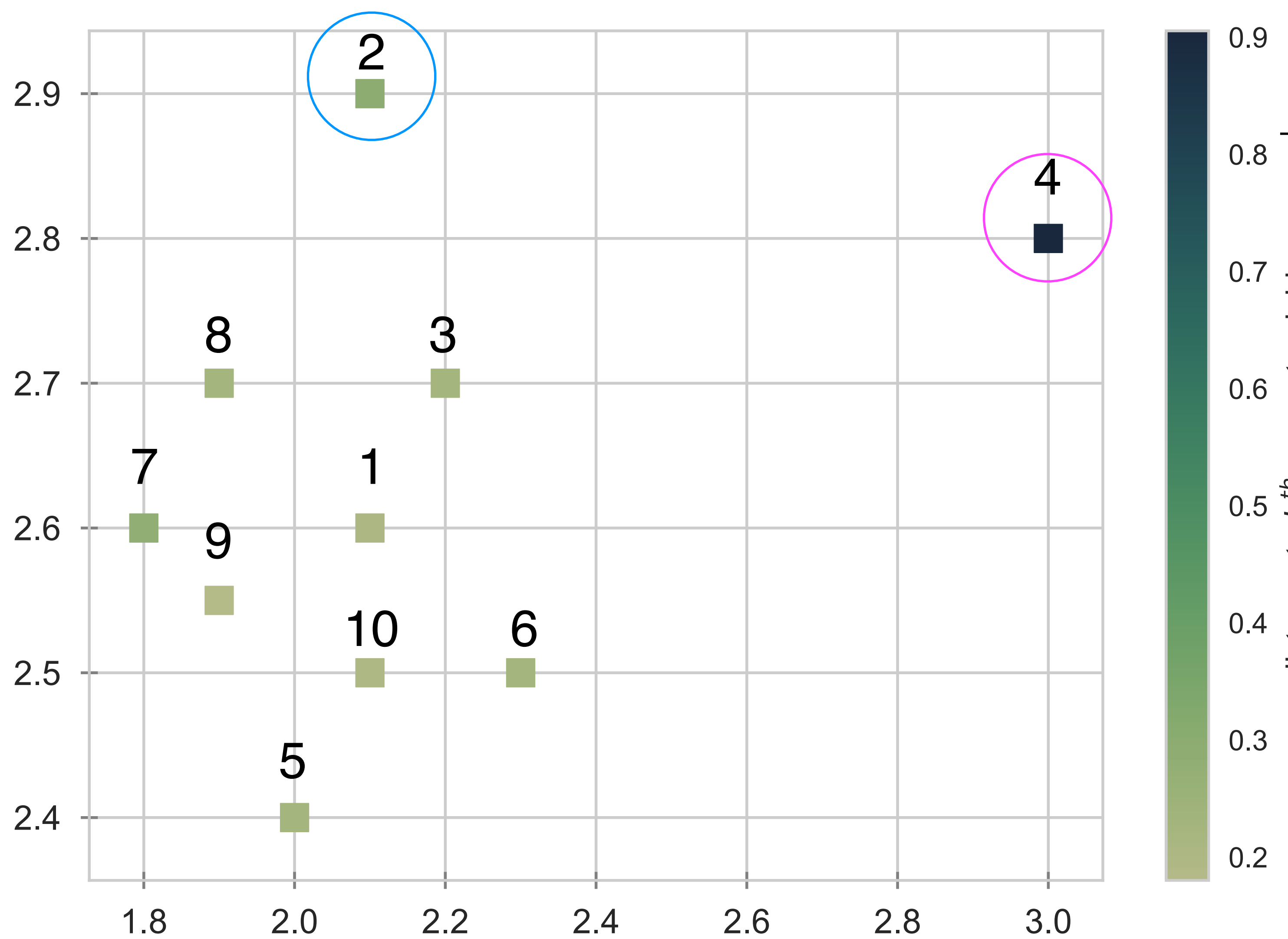


distances to *k*-NN

k=3	1-NN	2-NN	3-NN
1	0.10	0.14	0.21
2	0.22	0.28	0.30
3	0.14	0.22	0.22
4	0.76	0.81	0.91
5	0.14	0.18	0.22
6	0.20	0.22	0.22
7	0.11	0.14	0.28
8	0.14	0.15	0.22
9	0.11	0.15	0.18
10	0.10	0.14	0.20

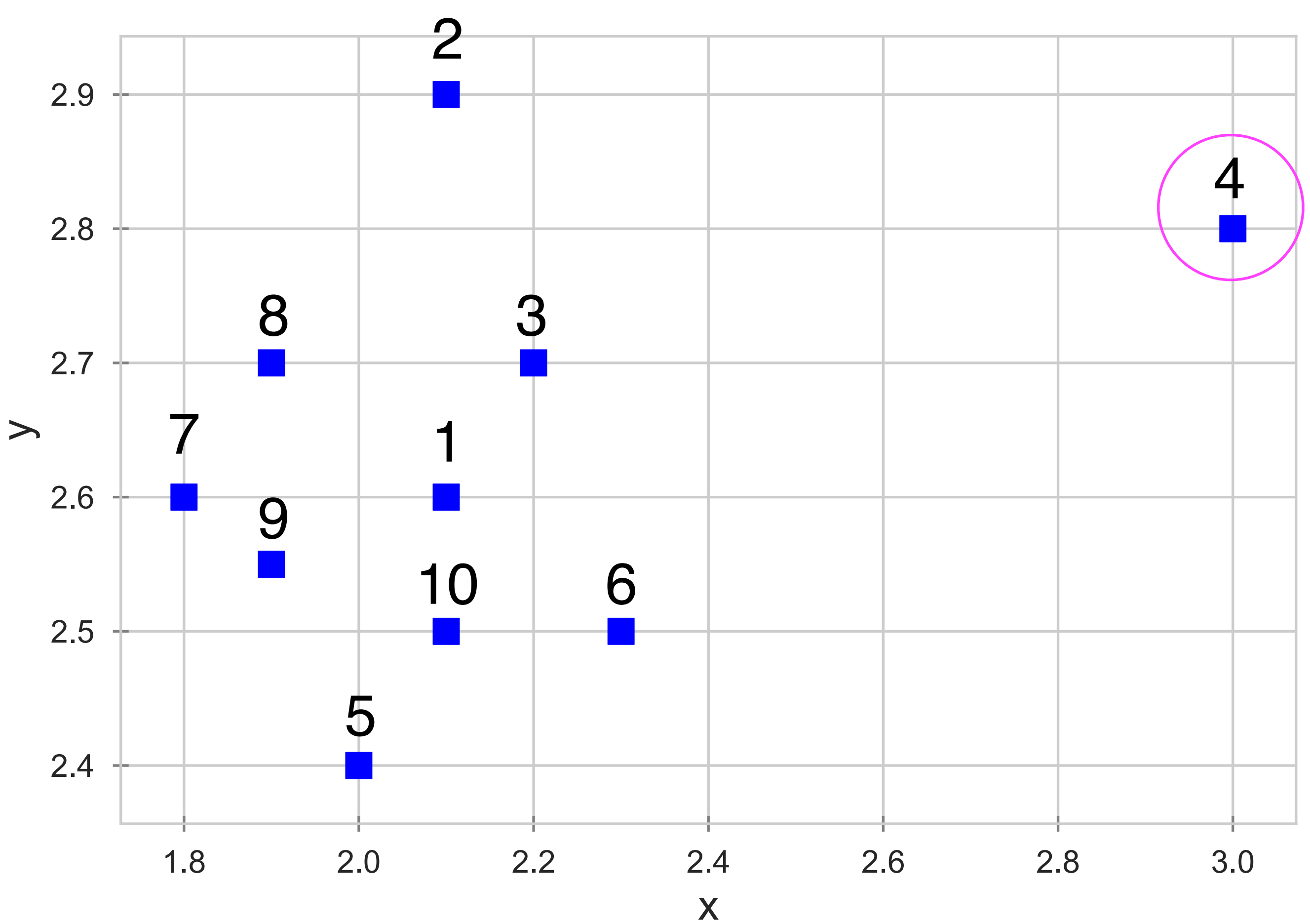
> Anomaly Detection / Distance-based

k=3



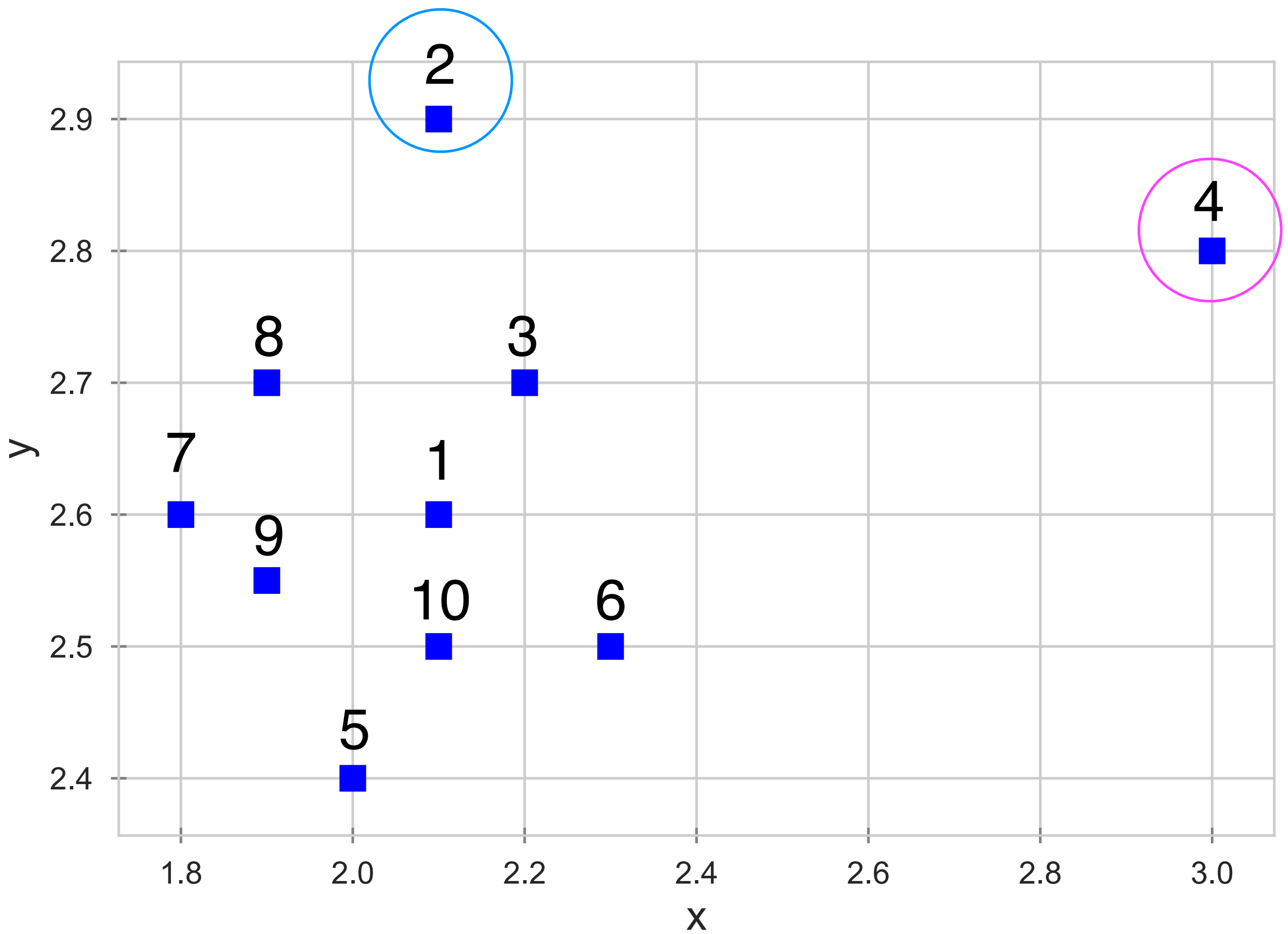
distances to *k*-NN

k=3	1-NN	2-NN	3-NN
1	0.10	0.14	0.21
2	0.22	0.28	0.30
3	0.14	0.22	0.22
4	0.76	0.81	0.91
5	0.14	0.18	0.22
6	0.20	0.22	0.22
7	0.11	0.14	0.28
8	0.14	0.15	0.22
9	0.11	0.15	0.18
10	0.10	0.14	0.20



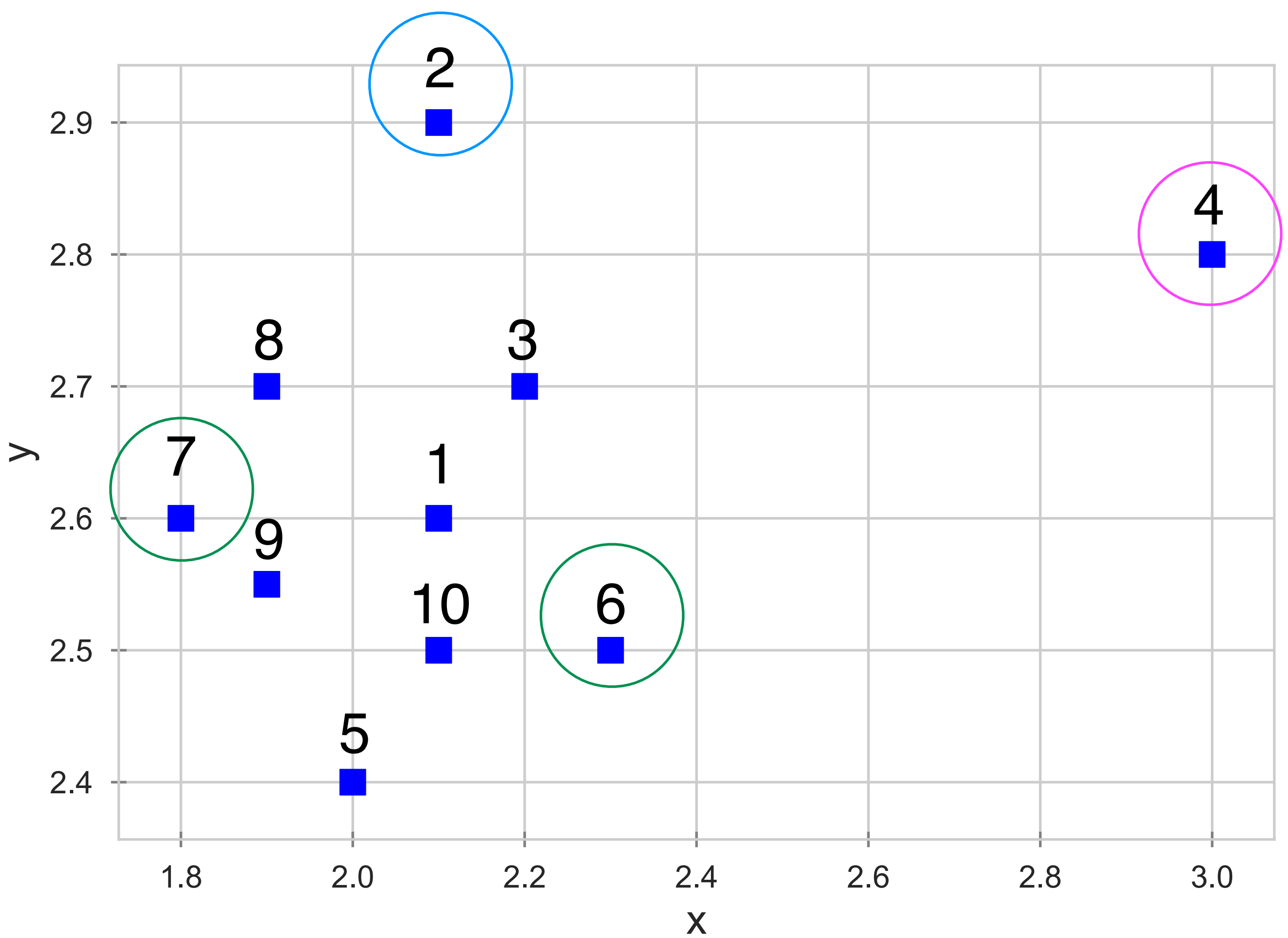
distances to k^{th}-NN

	k=1	*k=3*	*k=5*	*k=7*
1	0.1	0.21	0.22	0.30
2	0.22	0.30	0.40	0.45
3	0.14	0.22	0.30	0.36
4	0.76	0.91	0.95	1.10
5	0.14	0.22	0.32	0.36
6	0.20	0.22	0.40	0.45
7	0.11	0.28	0.32	0.42
8	0.14	0.22	0.28	0.32
9	0.11	0.18	0.21	0.40
10	0.10	0.20	0.22	0.32



distances to k^{th}-NN

	k=1	*k=3*	*k=5*	*k=7*
1	0.1	0.21	0.22	0.30
2	0.22	0.30	0.40	0.45
3	0.14	0.22	0.30	0.36
4	0.76	0.91	0.95	1.10
5	0.14	0.22	0.32	0.36
6	0.20	0.22	0.40	0.45
7	0.11	0.28	0.32	0.42
8	0.14	0.22	0.28	0.32
9	0.11	0.18	0.21	0.40
10	0.10	0.20	0.22	0.32



distances to k^{th}-NN

	k=1	k=3	k=5	k=7
1	0.1	0.21	0.22	0.30
2	0.22	0.30	0.40	0.45
3	0.14	0.22	0.30	0.36
4	0.76	0.91	0.95	1.10
5	0.14	0.22	0.32	0.36
6	0.20	0.22	0.40	0.45
7	0.11	0.28	0.32	0.42
8	0.14	0.22	0.28	0.32
9	0.11	0.18	0.21	0.40
10	0.10	0.20	0.22	0.32

Density-based approach

idea: density around an outlier is very different from the density around its neighbors

- define a density measure $d(x)$
- compare density around a point with the density around its local neighbors: relative density(x_i)
- compute *outlier score:* $S(x_i) = f($ relative density$(x_i)\ )$, for any x_i
- if $S(x_i) > threshold$, then x_i is an outlier

✓ can detect local anomalies
✓ good with variable densities
✗ complexity typically ~ $O(n^2)$
✗ sensitive to parameter choices

Local Outlier Factor (LOF)

$$\text{density}_k(p) = \left[\frac{1}{|N_k(p)|} \sum_{q \in N_k(p)} \text{dist}_k(p, q) \right]^{-1}$$

$|N_k(p)|$: n. of points around *p* within distance to its k-th nearest neighbor

$$\text{relative-density}_k(p) = \frac{\text{density}_k(p)}{\frac{1}{|N_k(p)|} \sum_{q \in N_k(p)} \text{density}_k(q)}$$

$$\text{LOF}_k(p) = \frac{1}{\text{relative-density}_k(p)} = \frac{1}{|N_k(p)|} \sum_{q \in N_k(p)} \frac{\text{density}_k(q)}{\text{density}_k(p)}$$

outlier score

LOF(*p*)>> 1 $\longrightarrow$ *p* is an outlier

- Suppose we have a **Validation Set** where some labels are known (anomaly/not anomaly)
- Run algo on Validation Set
- Evaluate metrics (confusion matrix, precision/recall, F-score)
- Choose parameters (epsilon, K, threshold, etc.) maximizing performance

it looks like supervised classification
with highly unbalanced classes

Confusion matrix

(for binary classification)

		actual class	
		1	0
predicted class	1	True Positive	False Positive
	0	False Negative	True Negative

precision = $\frac{\text{TP}}{\boxed{\text{TP+FP}}}$

predicted positives

recall = $\frac{\text{TP}}{\boxed{\text{TP+FN}}}$

actual positives

Trade-off precision/recall:

F-score = $\frac{2}{\frac{1}{P}+\frac{1}{R}} = 2\frac{PR}{P+R}$

3

DIMENSIONALITY REDUCTION

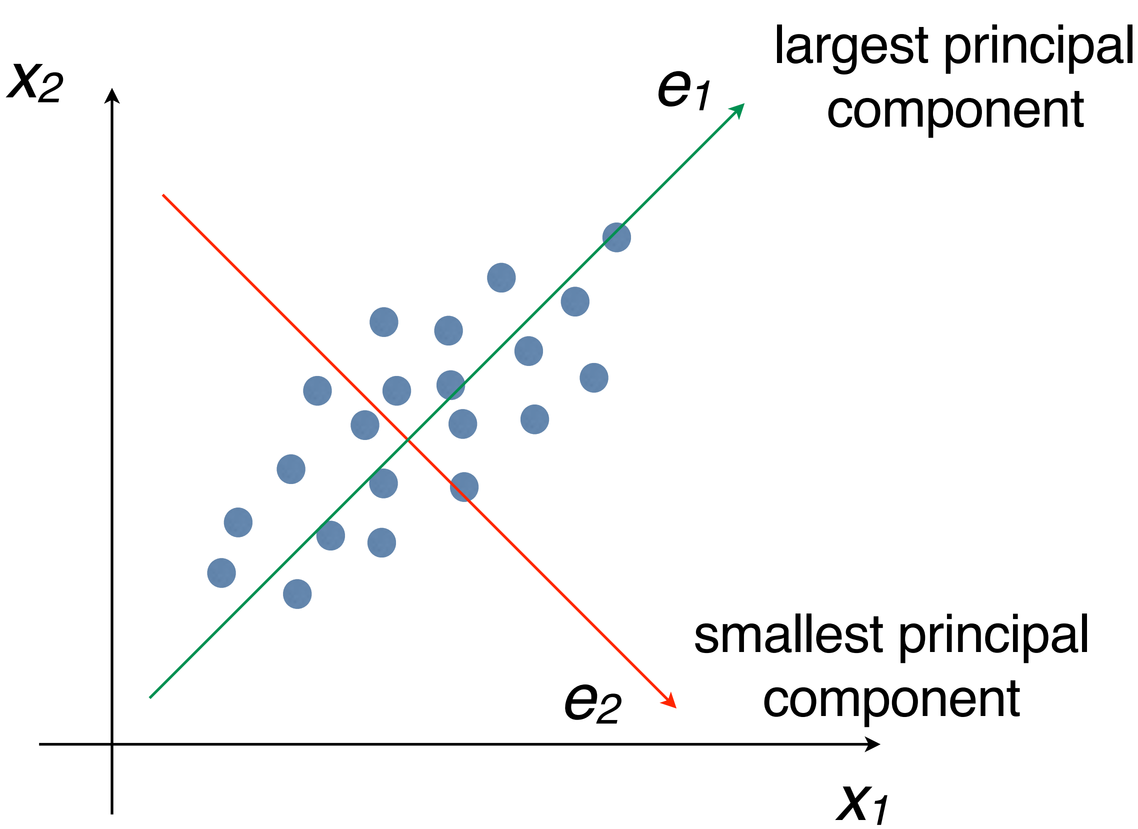
Approaches to Dimensionality Reduction

- Feature selection
  - choose subset of relevant features
  - Recursive Feature Elimination (RFE) + many others

- Feature projection
  - transform features into lower dimensional space
  - Principal Component Analysis (PCA) + many others

Principal Component Analysis

- PCA projects the entire dataset onto a different feature (sub)space, to maximize the variance
- reduce dimensions from *d* to *k*, retaining most of the info
- the largest princ. comp. is the direction of *greatest variability*



Why max variance?

- distant points in (x_1,x_2) are also distant in (e_1,e_2)
- minimize distances between original and projected points

- projected coordinates: $X' = X \cdot e$ *X: Nxd, e: dx1*
- maximize the variance of projections: $\max_e \frac{1}{N}(X \cdot e)^T (X \cdot e)\Big|_{e^T e=1}$
- implement constraint by Lagrange multiplier:

$$\max_e \frac{1}{N}(X \cdot e)^T (X \cdot e)\Big|_{e^T e=1} = \max_e L$$

$$L = \frac{1}{N}(X \cdot e)^T (X \cdot e) - \lambda(e^T e - 1)$$

- minimize *L*: $0 = \frac{\partial L}{\partial e^T} = \frac{1}{N}(X^T X e) - \lambda e$
- same as eigenvalue equation: $\Sigma e = \lambda e$

covariance matrix

$$\Sigma \equiv \frac{1}{N} X^T \cdot X$$

the directions of max variance are the eigenvectors of the cov. matrix

> Dimensionality Reduction / PCA

- **Input:** set of *N* examples (points), $\{x^{(1)}, \dots, x^{(N)}\}$ $\quad x^{(i)} \in \mathbb{R}^d$
- **Input:** number of principal components *k*

1. normalize data (zero mean, unit variance) $x^{(i)} \to \dfrac{x^{(i)} - \text{avg}(x)}{\text{std}(x)}$

2. compute *dxd* covariance matrix: $\Sigma = \dfrac{1}{N} X^T \cdot X = \dfrac{1}{N} \sum_{i=1}^{N} (x^{(i)})^T (x^{(i)})$
   X: Nxd

3. Diagonalize covariance and find eigenvectors: $\Sigma = V \Sigma_{\text{diag}} V^T$
   V*: dxd*

4. projection matrix *W* by truncating *V* to the first *k* columns
   (top *k* eigenvalues): $W = V(:, 1:k)$
   W: dxk

5. Project data along principal components: $Z = X \cdot W$
   Z*: Nxk*

- **Output:** new projected features *Z* living on *k*-dim space

Aside on SVD:

- Step 3 is typically replaced by Singular Value Decomposition (SVD) of X

$$X = U \cdot D \cdot V^T$$

- *U (Nxd)*: left singular vectors
  V (dxd): right singular vectors
  D (dxd): diagonal matrix of singular values of X

- The singular values of X are the square roots of the non-zero eigenvalues of both $X^T X$ and $X X^T$.

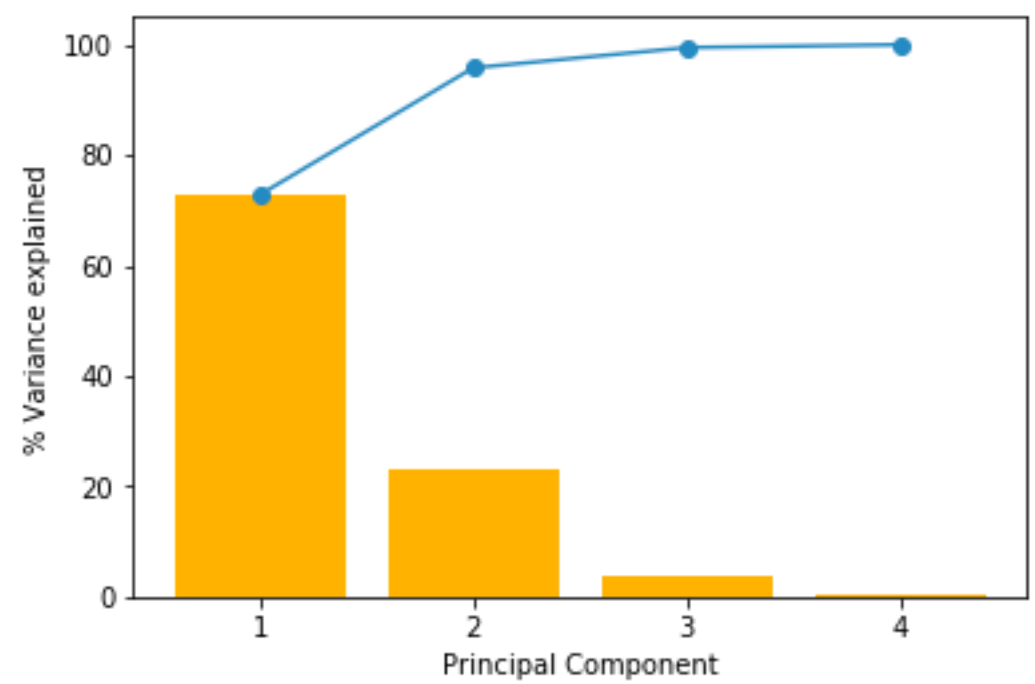
- No need to compute $X^T X$, better performance

- reconstruct original features $X_{\text{approx}} = Z \cdot W^T$
- error (as fraction of variance): $\dfrac{\sum_{i=1}^{N} |x^{(i)} - x^{(i)}_{\text{approx}}|^2}{\sum_{i=1}^{N} |x^{(i)}|^2} = 1 - \dfrac{\sum_{j=1}^{k} (\Sigma_{\text{diag}})_{jj}}{\sum_{j=1}^{d} (\Sigma_{\text{diag}})_{jj}}$

where
$$\Sigma_{\text{diag}} = \begin{pmatrix} \sigma_1^2 & & & & & \\ & \sigma_2^2 & & & 0 & \\ & & \ddots & & & \\ & & & \sigma_k^2 & & \\ & 0 & & & \ddots & \\ & & & & & \sigma_n^2 \end{pmatrix} \qquad \sigma_1^2 \geq \sigma_2^2 \geq \ldots \geq \sigma_n^2$$

- so want to maximize $R \equiv \dfrac{\sum_{j=1}^{k} (\Sigma_{\text{diag}})_{jj}}{\sum_{j=1}^{d} (\Sigma_{\text{diag}})_{jj}}$ (“**variance explained**”)

$R \geq 0.99$ “99% of variance explained”



> Dimensionality Reduction / PCA

- ✓ intuitive
- ✓ very general (applicable to ~ every dataset)
- ✓ good for coarse-grained classes
- ✘ only linear
- ✘ expensive (complexity O(N^3,D^3))
- ✘ poor for fine details

> Conclusions

- Unsupervised learning:
  tool to understand un-labelled data

- Look for Structures/Anomalies/Representations
  etc. of data

- Very active field of research

- Lots of applications yet to be explored