Data Science

Unsupervised Learning – Part 2

Lionel Fillatre

Polytech Nice Sophia

lionel.fillatre@univ-cotedazur.fr

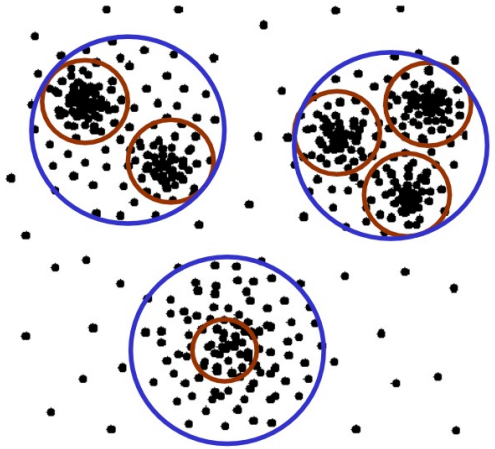
Outlines

- Course Logistics
- General Introduction
- What is unsupervised learning?
- Covariance Estimate
- Density Estimate
- Clustering
  - K-means
  - DBSCAN
  - Hierarchical clustering
  - Performance Metrics
- Gaussian mixture models
- Decomposing signals in components
  - PCA
  - Kernel PCA
- Novelty and outlier detection
  - Robust covariance
  - One-class SVM
  - Isolation forest
  - Local outlier factor
- Manifold learning
  - Locally Linear Embedding
- Conclusion

Hierarchical clustering

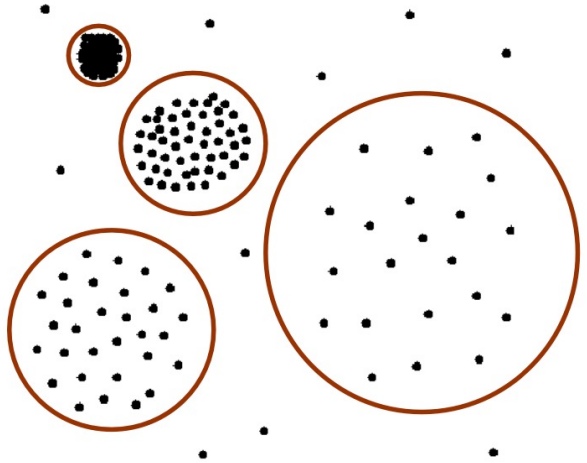
Problematic example

- Global parameters to separate all clusters with a partitioning clustering method may not exist
- Need a hierarchical clustering algorithm in these situations



hierarchical cluster structure

and/or

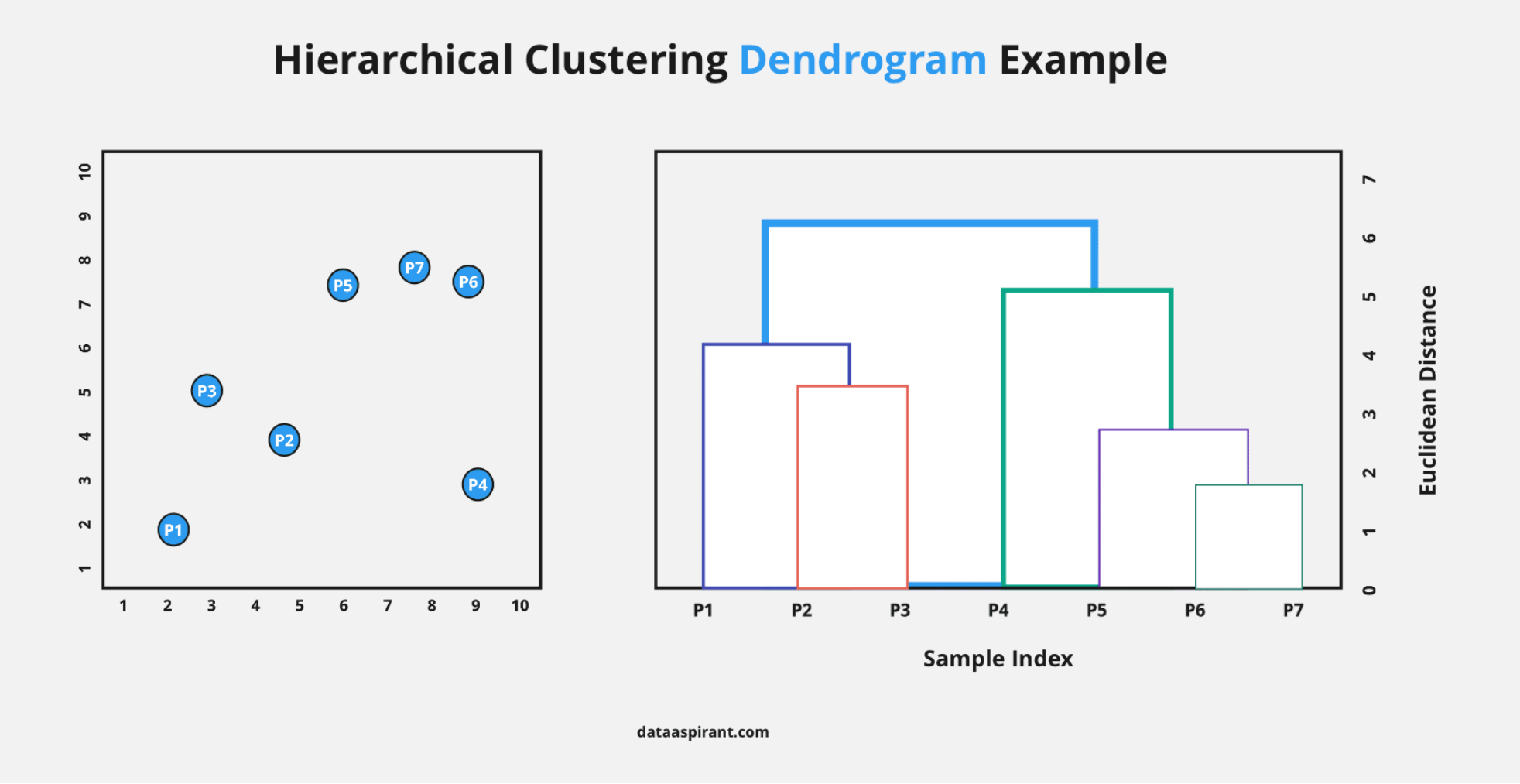


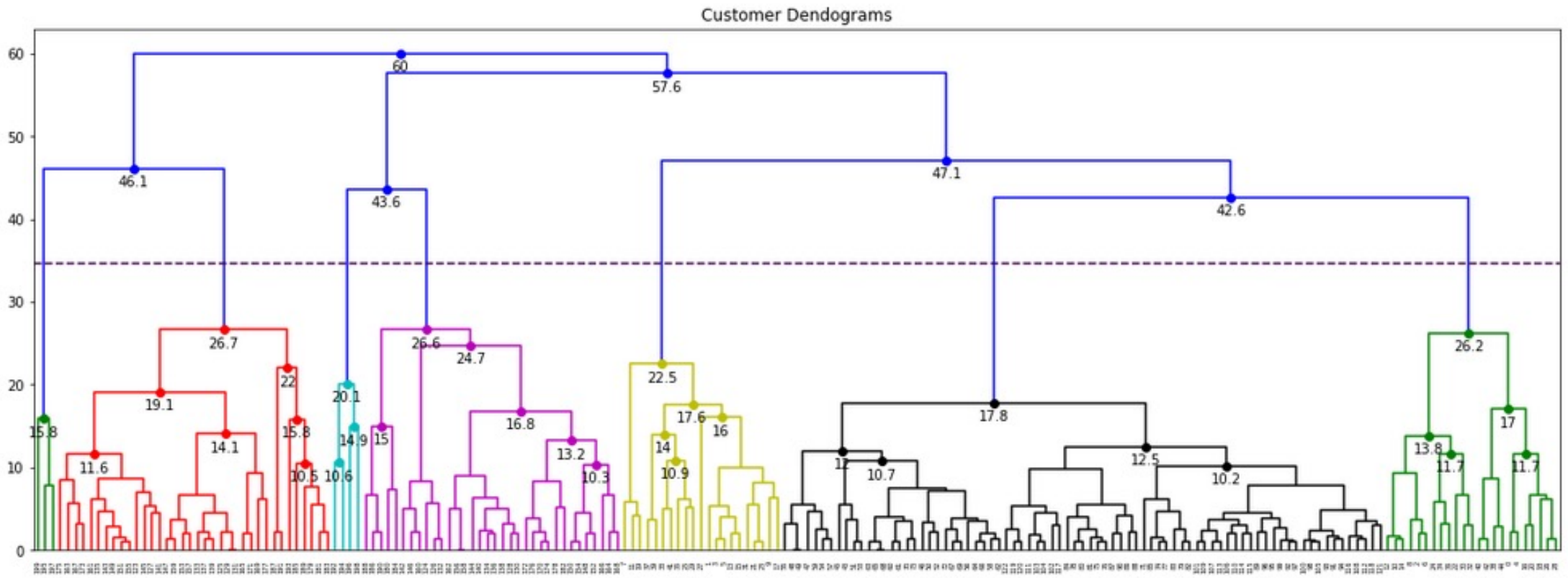
largely differing densities and sizes

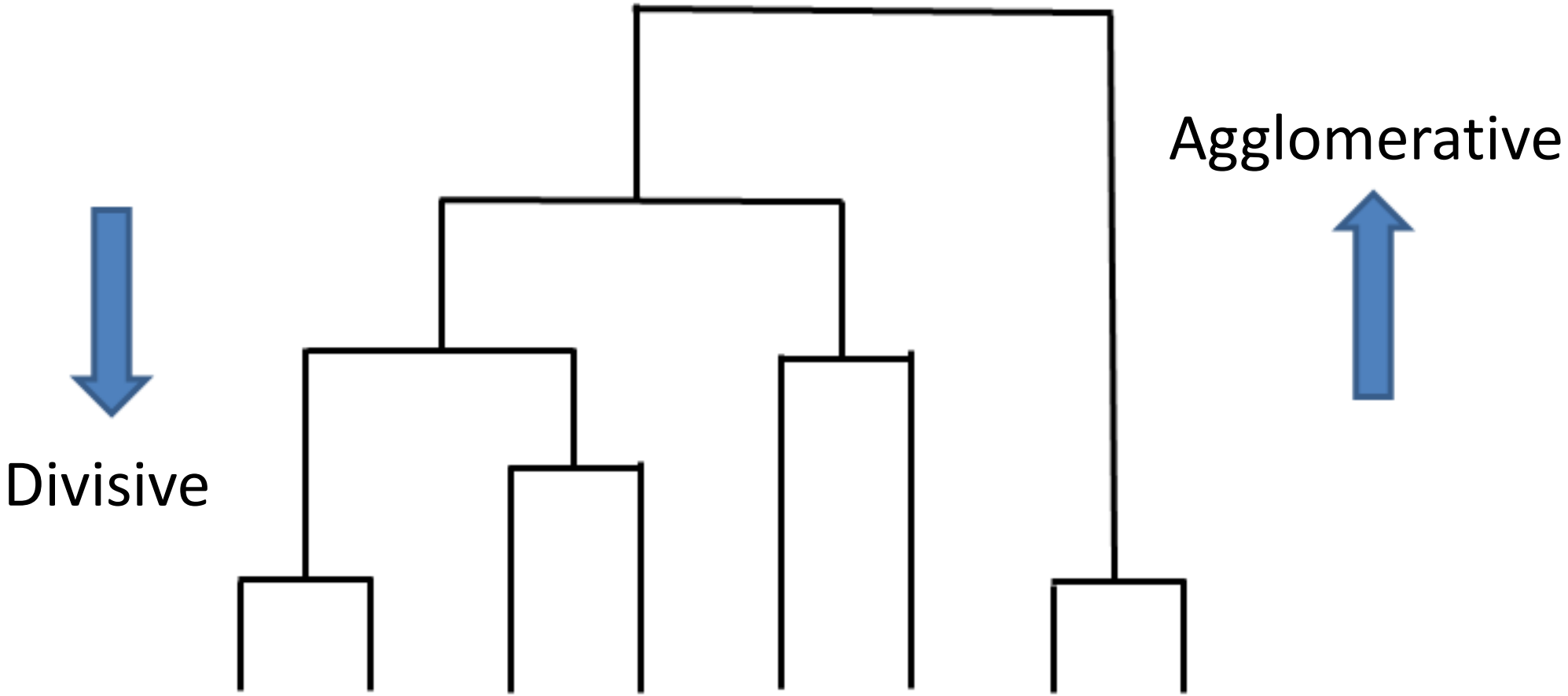
Hierarchical clustering

- In practice, we might be interested in learning a hierarchical clustering, where clusters are nested inside larger clusters
- There are two main approaches to hierarchical clustering
  - Bottom-up: agglomerative clustering builds up a hierarchy by starting with a fine-grain clustering and merging clusters together
  - Top-down: divisive clustering builds up a hierarchy by iteratively dividing existing clusters into smaller components

Dendogram


Hierarchical Clustering Dendrogram Example
P1
P2
P3
P4
P5
P6
P7
Euclidean Distance
Sample Index
dataaspirant.com


Customer Dendograms
60
57.6
46.1
43.6
47.1
42.6
26.7
26.6
24.7
22
20.1
22.5
17.6
16
17.8
26.2
19.1
16.8
14.1
15.8
14.9
15
14
10.9
13.2
10.3
10.5
10.6
12.5
10.2
13.8
11.7
17
11.7
11.6
10.7
15.8
0
10
20
30
40
50
60


Agglomerative
Divisive

Agglomerative clustering

- Initialize by defining each sample to be its own cluster
- At each iteration
  - Evaluate some metric measuring how similar each possible pair of clusters are (linkage methods)
  - Merge the most similar pair of clusters
  - Repeat until all clusters are merged

https://uc-r.github.io/hc_clustering

Linkage methods

- Suppose that $\mathcal{A}$ and $\mathcal{B}$ represent the index sets for two clusters which are candidates for being merged
- Let $D(\boldsymbol{x}, \boldsymbol{y})$ be a distance (or metric) between two single points $\boldsymbol{x}$ and $\boldsymbol{y}$
  - Example: $D(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|_2$
- How can we measure how similar these two clusters $\mathcal{A}$ and $\mathcal{B}$ are?
  - Nearest neighbor (single linkage clustering): $\min_{i \in \mathcal{A}, j \in \mathcal{B}} D(\boldsymbol{x}_i, \boldsymbol{x}_j)$
    - Tends to produce long chains
  - Farthest neighbor (complete linkage clustering): $\max_{i \in \mathcal{A}, j \in \mathcal{B}} D(\boldsymbol{x}_i, \boldsymbol{x}_j)$
    - Tends to produce « spherical » clusters
  - Average (average linkage clustering): $\frac{1}{|\mathcal{A}||\mathcal{B}|} \sum_{i \in \mathcal{A}} \sum_{j \in \mathcal{B}} D(\boldsymbol{x}_i, \boldsymbol{x}_j)$
    - Less affected by outliers

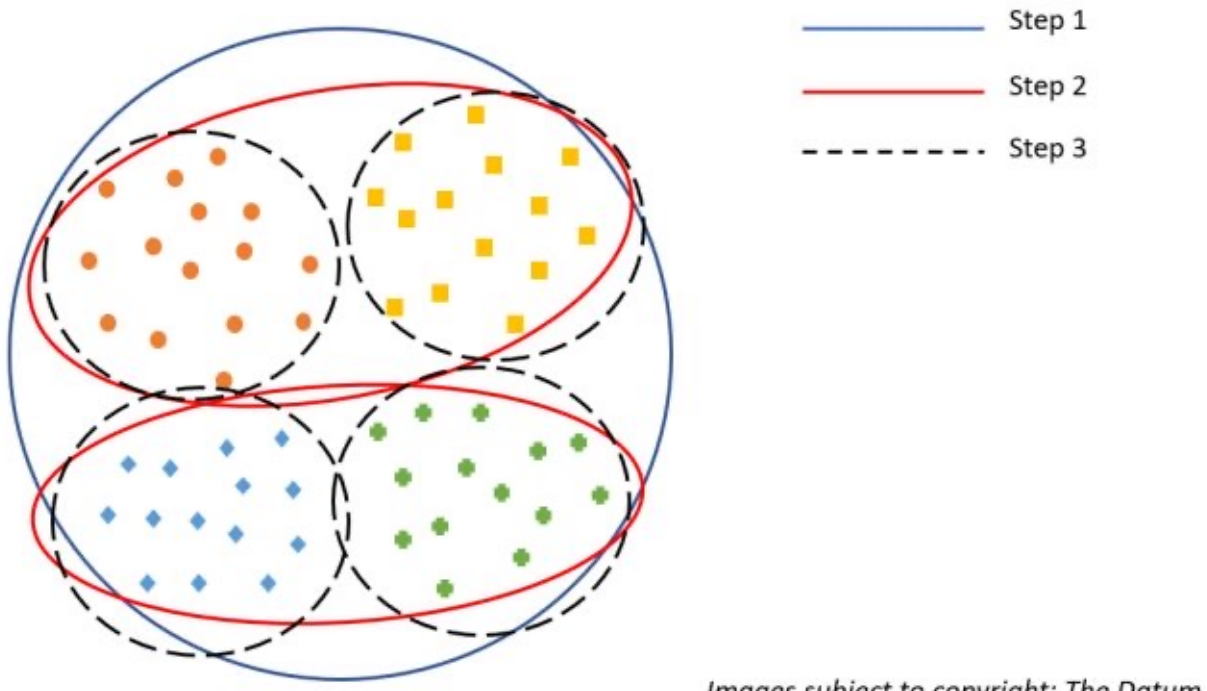
Single-linkage Clustering Algorithm

- **Notations :**
  - $N \times N$ proximity matrix D that contains all distances $D(\boldsymbol{x}_i, \boldsymbol{x}_j)$ where N is initially the number of samples
  - The clusterings are assigned sequence numbers $0, 1, 2, \ldots, n-1$
  - A cluster with sequence number m is denoted $\langle m \rangle$
  - $L(k)$ is the level of the k-th clustering (it is a real value that depends on to the distance between the samples).
  - The proximity between clusters $\langle r \rangle$ and $\langle s \rangle$ is denoted $D(\langle r \rangle, \langle s \rangle)$
- **Algorithm:**
  1. Begin with the disjoint clustering, i.e. $\langle r \rangle = \{r\}$ for all r, having level $L(0) = 0$ and sequence number $m = 0$.
  2. Find the most similar pair of clusters in the current clustering, say pair $\langle r \rangle$ and $\langle s \rangle$ according to $D(\langle r \rangle, \langle s \rangle) = \min_{\langle i \rangle, \langle j \rangle} D(\langle i \rangle, \langle j \rangle)$ where the minimum is over all pairs of clusters in the current clustering.
  3. Increment the sequence number: $m = m + 1$.
  4. Merge clusters $\langle r \rangle$ and $\langle s \rangle$ into a single cluster to form the next clustering m.
  5. Set the level of this clustering to $L(m) = D(\langle r \rangle, \langle s \rangle)$
  6. Update the proximity matrix, D, by deleting the rows and columns corresponding to clusters $\langle r \rangle$ and $\langle s \rangle$ and adding a row and column corresponding to the newly formed cluster. The proximity between the new cluster, denoted $\langle r, s \rangle$ and an old cluster $\langle k \rangle$ is defined as
  $$D(\langle r, s \rangle, \langle k \rangle) = \min\{D(\langle k \rangle, \langle r \rangle), D(\langle k \rangle, \langle s \rangle)\}$$
  7. If all objects are in one cluster, stop. Else, go to step 2.

Divisive clustering

- Begin with the entire dataset belonging to a single cluster
- At each iteration, need to
  - Select an existing cluster to divide (can use a metric like within-cluster scatter)
  - Divide the selected cluster into two child clusters (using any clustering algorithm, e.g., kmeans)
- Can be computationally demanding, and is somewhat less well studied than agglomerative clustering

DIvisive ANAlysis (DIANA)



DIvisive ANAlysis (DIANA)

- Let $C_0 = \{1, \dots, N\}$ be the set of all N object indices and $\mathcal{C} = \{C_0\}$ the set of all formed clusters so far.
- Iterate the following until $|\mathcal{C}| = N$:
  1. Find the current cluster with 2 or more objects that has the largest diameter: $C^* = \underset{C \in \mathcal{C}}{\operatorname{argmax}} \max_{i_1, i_2 \in C} D\left(\boldsymbol{x}_{i_1}, \boldsymbol{x}_{i_2}\right)$
  2. Find the object in this cluster with the most dissimilarity to the rest of the cluster:
  $$i^* = \underset{i \in C^*}{\operatorname{argmax}} \frac{1}{|C^*| - 1} \sum_{j \in C^* \setminus \{i\}} D\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right)$$
  3. Pop i^* from its old cluster C^* and put it into a new splinter group $C_{new} = \{i^*\}$
  4. Keep migrating objects from C^* to add them to C_{new}. To choose which objects to migrate, don't just consider dissimilarity to C^*, but also adjust for dissimilarity to the splinter group:
  $$i^* = \underset{i \in C^*}{\operatorname{argmax}}\, D(i) \qquad \text{where} \qquad D(i) = \frac{1}{|C^*| - 1} \sum_{j \in C^* \setminus \{i\}} D\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) - \frac{1}{|C_{new}|} \sum_{j \in C_{new}} D\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right)$$
  then either stop iterating when $D(i^*) < 0$, or migrate i^*.
  5. Add C_{new} to $\mathcal{C}$.

https://en.wikipedia.org/wiki/Hierarchical_clustering

Performance Metrics

Evaluation of Clustering Results

- Evaluation based on expert's opinion
  + may reveal new insight into the data
  – very expensive, results are not comparable

- Evaluation based on internal measures
  + no additional information needed
  – approaches optimizing the evaluation criteria will always be preferred

- Evaluation based on external measures
  + objective evaluation
  – needs « ground truth »

https://en.wikipedia.org/wiki/Cluster_analysis

Evaluation based on internal measures

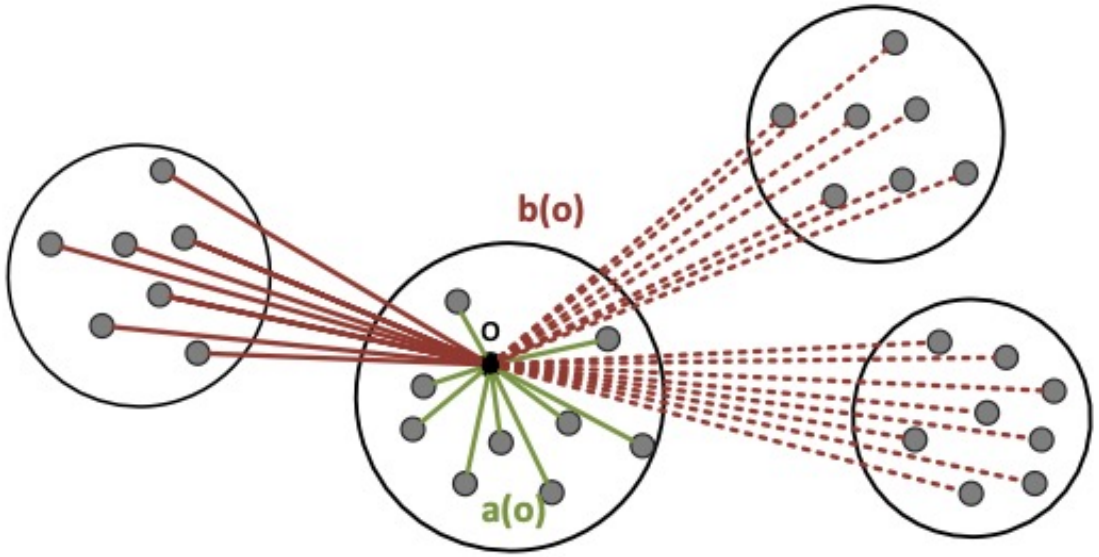
- Given a clustering $\mathcal{C} = \{C_1, \dots, C_k\}$ for a dataset composed of N points
- Cohesion: measures the similarity of objects within a cluster
- Separation: measures the dissimilarity of one cluster to another one
- Most famous measure (this does not mean the « best »!)
  - Sum of square distances:

$$SSD(\mathcal{C}) = \frac{1}{N} \sum_{C_i \in \mathcal{C}} \sum_{\boldsymbol{x} \in C_i} dist\big(\boldsymbol{x}, \mu(C_i)\big)^2$$

  - $\mu(C_i)$ is a centroid of cluster C_i

The Silhouette coefficient (1)

- How good is the clustering = how appropriate is the mapping of objects to clusters
- Elements in cluster should be « similar » to their representative
  - Measure the average distance of points to their representative: $a(\boldsymbol{x})$
- Elements in different clusters should be « dissimilar »
  - Measure the average distance of points to alternative clusters (i.e. second closest cluster): $b(\boldsymbol{x})$



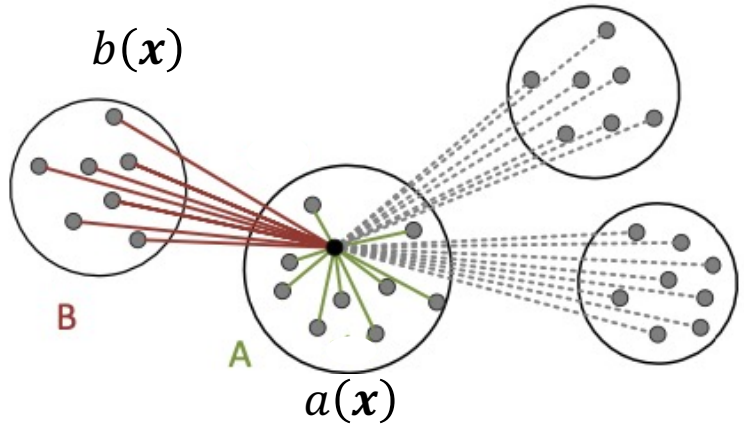
The Silhouette coefficient (2)

- $a(\boldsymbol{x})$: average distance between object $\boldsymbol{x}$ and the objects in its cluster $A = C(\boldsymbol{x})$

$$a(x) = \frac{1}{|C(x)|} \sum_{p \in C(\boldsymbol{x})} dist(\boldsymbol{x}, p)$$

- $b(\boldsymbol{x})$: for each other cluster C_i compute the average distance between $\boldsymbol{x}$ and the objects in C_i. Then take the smallest average distance

$$b(\boldsymbol{x}) = \min_{C_i \neq C(\boldsymbol{x})} \left(\frac{1}{|C_i|} \sum_{p \in C_i} dist(\boldsymbol{x}, p) \right)$$



- The silhouette of $\boldsymbol{x}$ is then defined as

$$s(\boldsymbol{x}) = \begin{cases} 0 & if\ a(\boldsymbol{x}) = 0 \\ \dfrac{b(\boldsymbol{x}) - a(\boldsymbol{x})}{\max\{a(\boldsymbol{x}), b(\boldsymbol{x})\}} & else \end{cases}$$

- The values of the silhouette coefficient range from –1 to +1

The Silhouette coefficient (3)

- The silhouette of a cluster C_i is defined as:

$$silh(C_i) = \frac{1}{|C_i|} \sum_{\boldsymbol{x} \in C_i} s(\boldsymbol{x})$$

- The silhouette of a clustering $\mathcal{C} = (C_1, \ldots, C_k)$ is defined a...

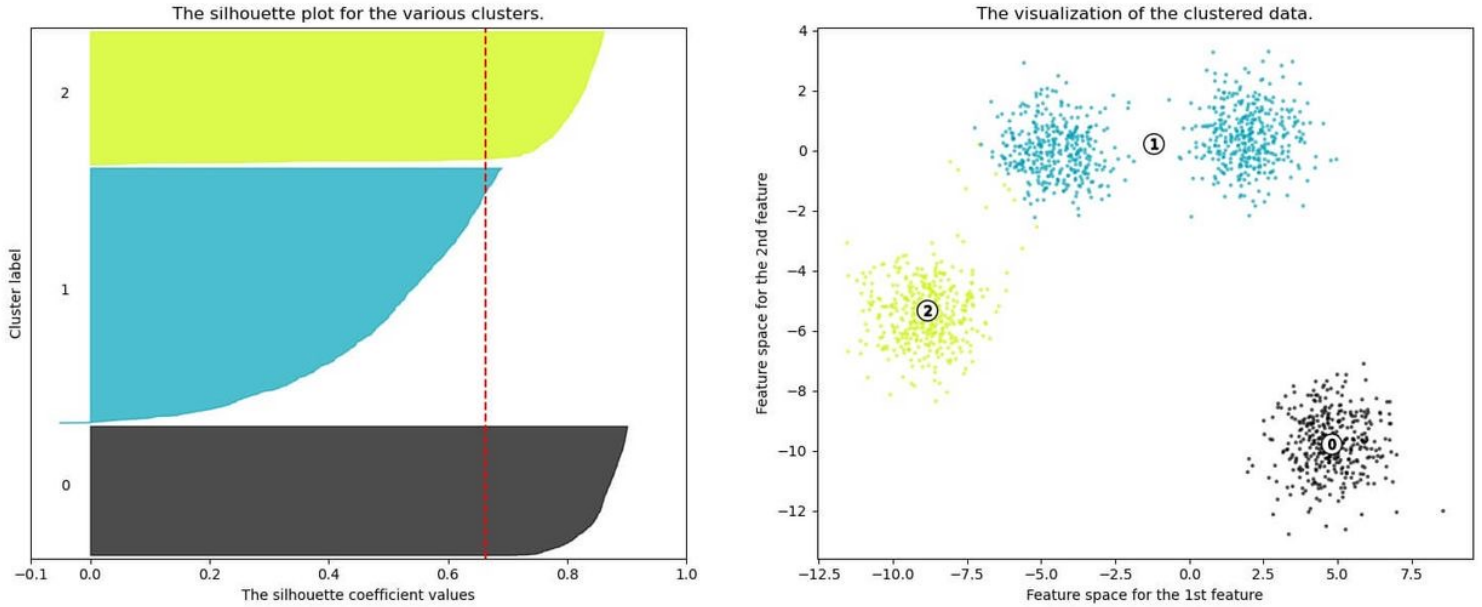
$$silh(\mathcal{C}) = \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x} \in \mathcal{D}} s(\boldsymbol{x})$$

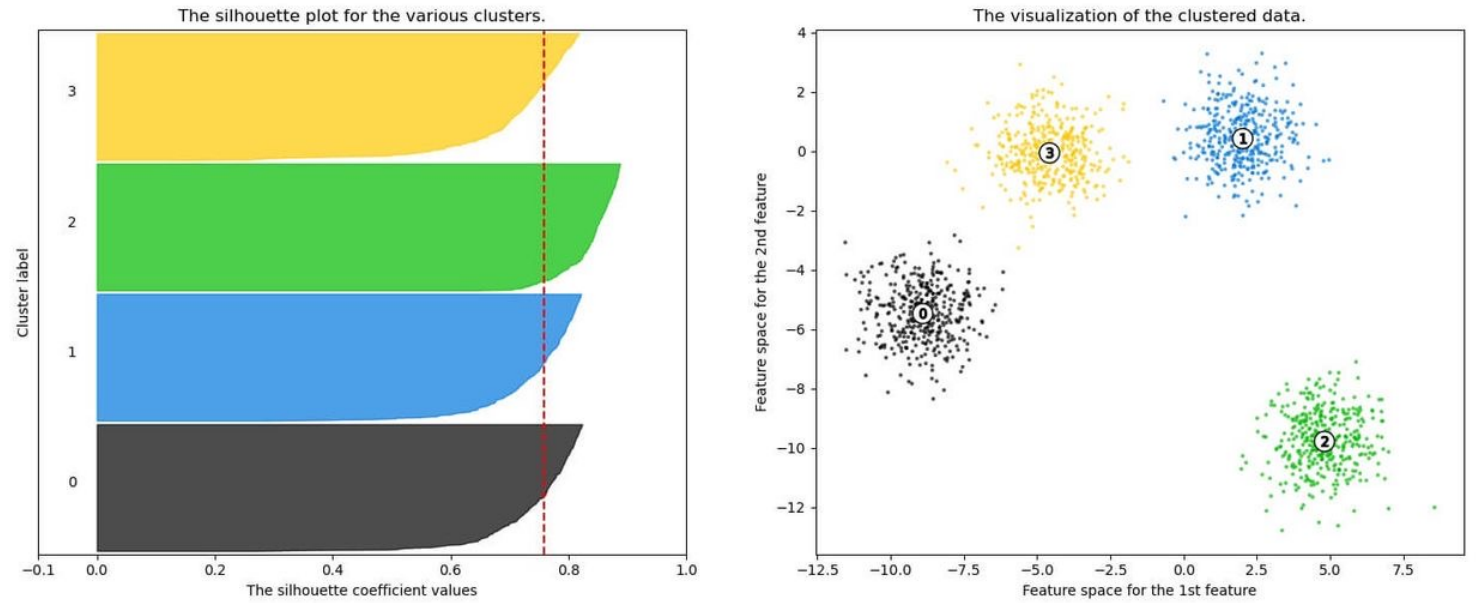
where $\mathcal{D}$ denotes the whole dataset

The Silhouette coefficient (4)

- « Reading » the silhouette coefficient:
- Let $a(\boldsymbol{x}) \neq 0$
  - $b(\boldsymbol{x}) \gg a(\boldsymbol{x}) \Rightarrow s(\boldsymbol{x}) \approx 1$: good assignment of o to its cluster $A = C(\boldsymbol{x})$
  - $b(\boldsymbol{x}) \approx a(\boldsymbol{x}) \Rightarrow s(\boldsymbol{x}) \approx 0$: $\boldsymbol{x}$ is in-between A and B
  - $b(\boldsymbol{x}) \ll a(\boldsymbol{x}) \Rightarrow s(\boldsymbol{x}) \approx -1$: bad, on average $\boldsymbol{x}$ is closer to members of B

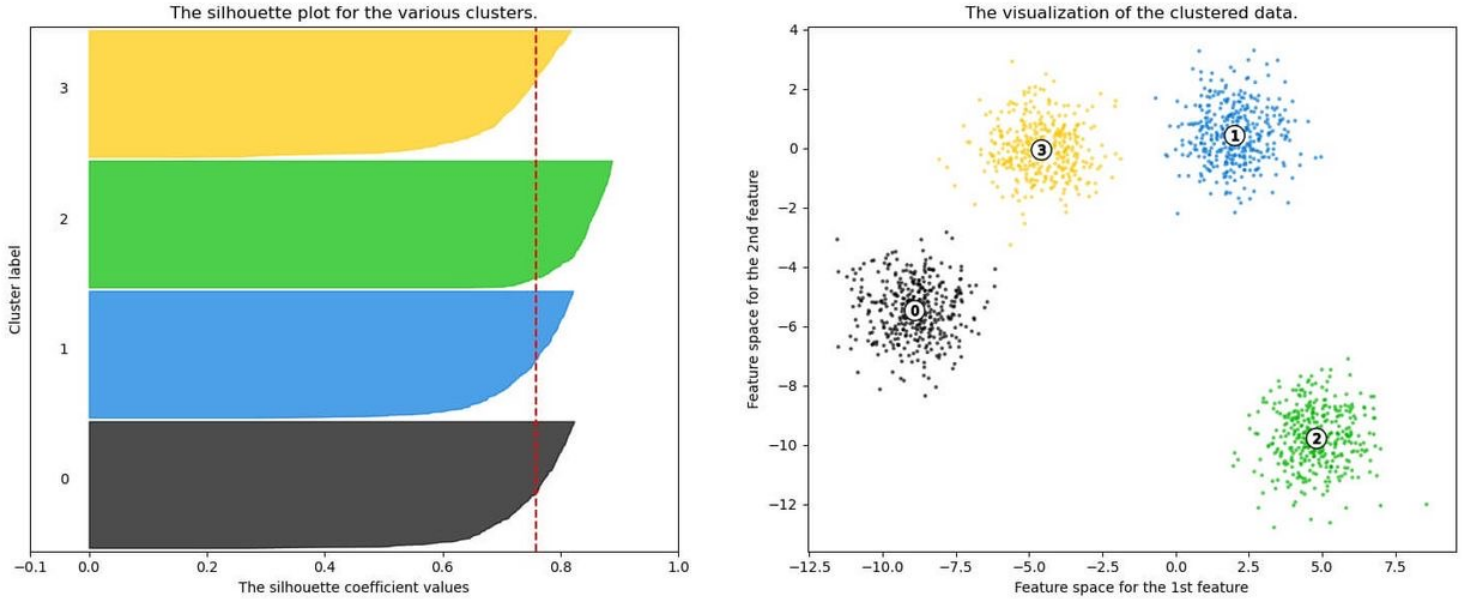
- Silhouette Coefficient $silh(\mathcal{C})$ of a clustering $\mathcal{C} = (C_1, \ldots, C_k)$:
  - average silhouette of all objects
    - $0.7 < silh(\mathcal{C}) < 1.0$ strong structure,
    - $0.5 < silh(\mathcal{C}) \leq 0.7$ medium structure
    - $0.25 < silh(\mathcal{C}) \leq 0.5$ weak structure,
    - $silh(\mathcal{C}) \leq 0.25$ no structure



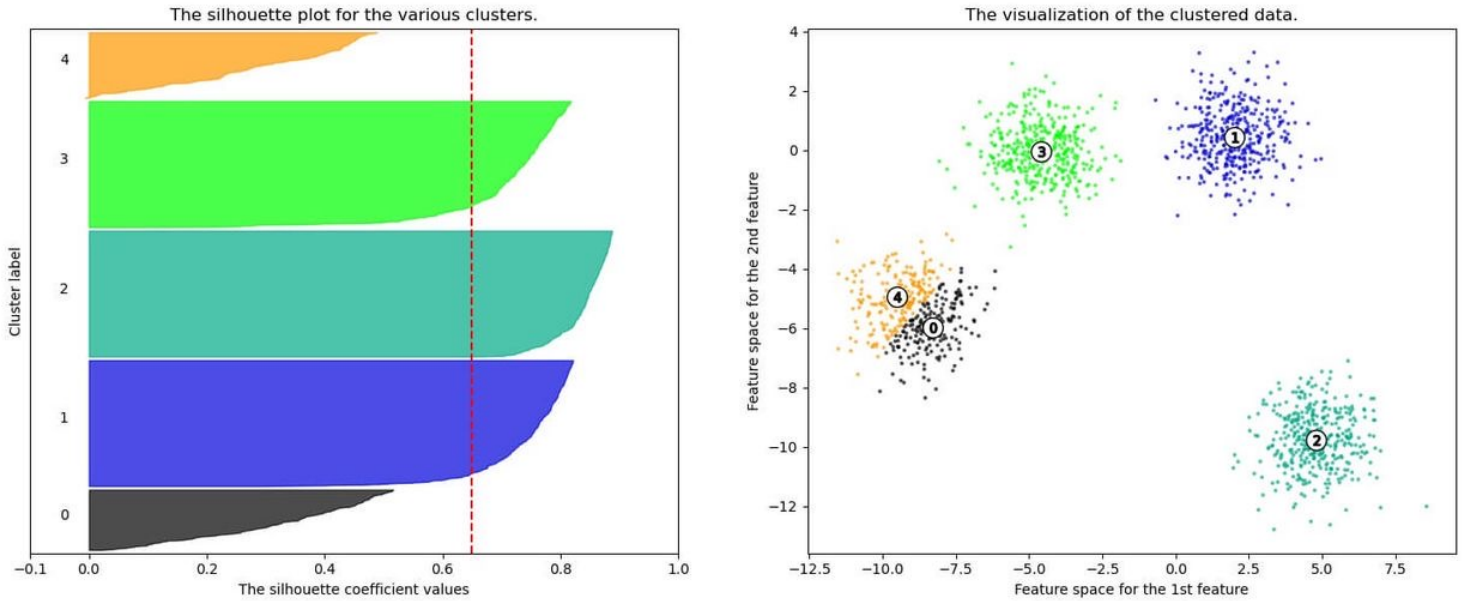


https://ml-explained.com/blog/kmeans-explained

Silhouette analysis for KMeans clustering on sample data with n_clusters = 4



Silhouette analysis for KMeans clustering on sample data with n_clusters = 5



Evaluation based on external measures (1)

- Given a clustering $\mathcal{C} = \{C_1, \dots, C_k\}$ and ground truth $\mathcal{G} = \{G_1, \dots, G_\ell\}$
- Recall: $rec(C_i, G_j) = \frac{|C_i \cap G_j|}{|G_j|}$
- Precision: $prec(C_i, G_j) = \frac{|C_i \cap G_j|}{|C_i|}$
- F-Measure: $F(\mathcal{C}) = \sum_{C_i \in \mathcal{C}} \frac{|C_i|}{|\mathcal{D}|} \max_{G_j \in \mathcal{G}} F(C_i, G_j)$ where $F(C_i, G_j) = \frac{2 * rec(C_i, G_j) * prec(C_i, G_j)}{rec(C_i, G_j) + prec(C_i, G_j)}$
- Purity: $P(\mathcal{C}, \mathcal{G}) = \sum_{C_i \in \mathcal{C}} \frac{|C_i|}{|\mathcal{D}|} pur(C_i, \mathcal{G})$ where $pur(C_i, \mathcal{G}) = \max_{G_j \in \mathcal{G}} prec(C_i, G_j)$

Evaluation based on external measures (2)

- Given a clustering $\mathcal{C} = \{C_1, \dots, C_k\}$ and ground truth $\mathcal{G} = \{G_1, \dots, G_\ell\}$
- Rand Index: $RI(\mathcal{C}, \mathcal{G}) = \frac{a+b}{a+b+c+d} = \frac{a+b}{\binom{N}{2}} = \frac{\text{Number.of pair-wise predictions}}{\text{Total number of possible pairs}}$
  - a: the number of pairs of elements in $\mathcal{D}$ that are in the same subset in $\mathcal{C}$ and in the same subset in $\mathcal{G}$

$$a = \{(o_i, o_j) \in \mathcal{D}\times\mathcal{D}: (o_i \neq o_j) \wedge (\exists C \in \mathcal{C}: o_i, o_j \in C) \wedge (\exists G \in \mathcal{G}: o_i, o_j \in G)\}$$

  - b: the number of pairs of elements in $\mathcal{D}$ that are in different subsets $\mathcal{C}$ and in different subsets in $\mathcal{G}$

$$b = \{(o_i, o_j) \in \mathcal{D}\times\mathcal{D}: (o_i \neq o_j) \wedge \neg(\exists C \in \mathcal{C}: o_i, o_j \in C) \wedge \neg(\exists G \in \mathcal{G}: o_i, o_j \in G)\}$$

  - c: the number of pairs of elements in $\mathcal{D}$ that are in the same subset in $\mathcal{C}$ and in different subsets in $\mathcal{G}$

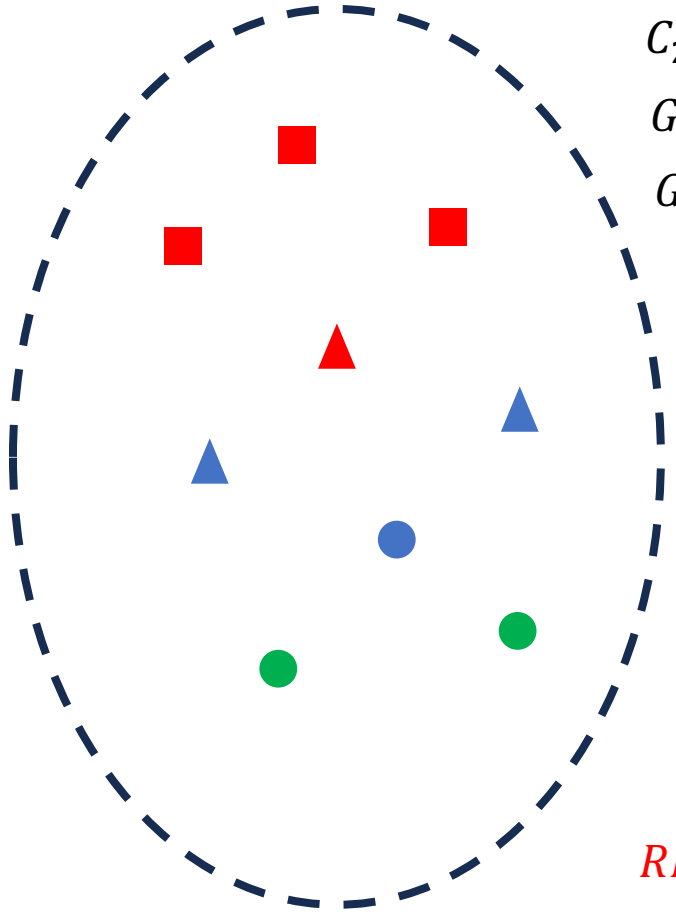
$$c = \{(o_i, o_j) \in \mathcal{D}\times\mathcal{D}: (o_i \neq o_j) \wedge (\exists C \in \mathcal{C}: o_i, o_j \in C) \wedge \neg(\exists G \in \mathcal{G}: o_i, o_j \in G)\}$$

  - d: the number of pairs of elements in $\mathcal{D}$ that are in different subsets in $\mathcal{C}$ and in the same subset in $\mathcal{G}$

$$d = \{(o_i, o_j) \in \mathcal{D}\times\mathcal{D}: (o_i \neq o_j) \wedge \neg(\exists C \in \mathcal{C}: o_i, o_j \in C) \wedge (\exists G \in \mathcal{G}: o_i, o_j \in G)\}$$

Illustration

$\mathcal{C} = \{C_1, C_2, C_3\}$

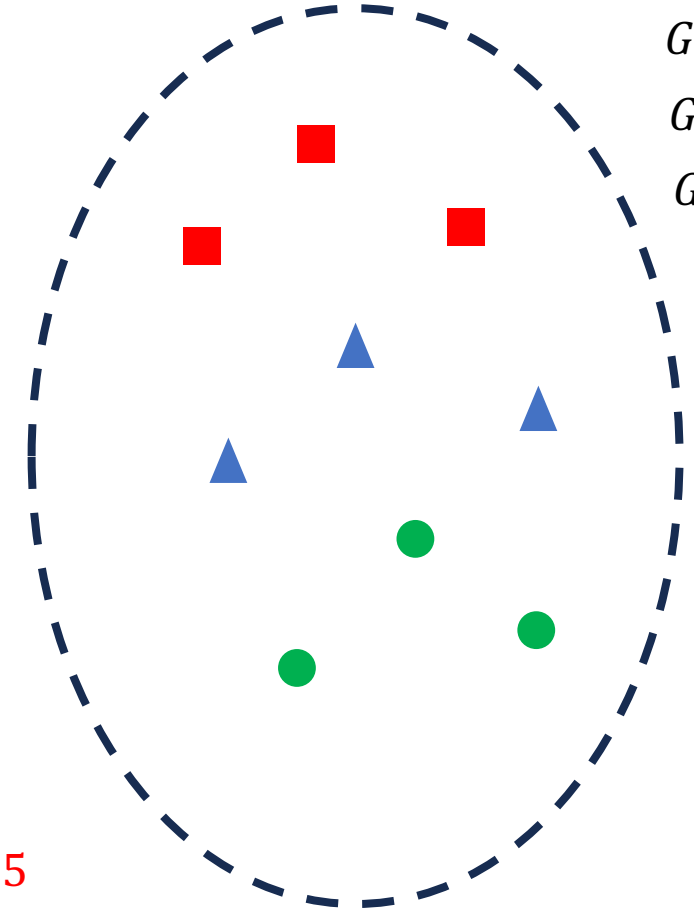


$C_2 =$ ■ ▲

$G_2 =$ ▲ ●

$G_3 =$ ●

$\mathcal{G} = \{G_1, G_2, G_3\}$



$G_1 =$ ■

$G_2 =$ ▲

$G_3 =$ ●

$$RI(\mathcal{C}, \mathcal{G}) = \frac{5 + 22}{36} = 0.75$$

Evaluation based on external measures (3)

- Given a clustering $\mathcal{C} = \{C_1, \dots, C_k\}$ and ground truth $\mathcal{G} = \{G_1, \dots, G_\ell\}$
- Mutual Entropy:

$$H(\mathcal{G}|\mathcal{C}) = -\sum_{C_i \in \mathcal{C}} p(C_i) \sum_{G_j \in \mathcal{G}} p\left(G_j|C_i\right) \log p\left(G_j|C_i\right)$$

$$H(\mathcal{G}|\mathcal{C}) = -\sum_{C_i \in \mathcal{C}} \frac{|C_i|}{N} \sum_{G_j \in \mathcal{G}} \frac{\left|C_i \cap G_j\right|}{|C_i|} \log \frac{\left|C_i \cap G_j\right|}{|C_i|}$$

- Mutual Information:

$$I(\mathcal{C}, \mathcal{G}) = H(\mathcal{C}) - H(\mathcal{C}|\mathcal{G}) = H(\mathcal{G}) - H(\mathcal{G}|\mathcal{C})$$

where entropy

$$H(\mathcal{C}) = -\sum_{C_i \in \mathcal{C}} p(C_i) \log p(C_i) = -\sum_{C_i \in \mathcal{C}} \frac{|C_i|}{N} \log \frac{|C_i|}{N}$$

- Normalized Mutual Information:

$$NMI(\mathcal{C}, \mathcal{G}) = \frac{I(\mathcal{C}, \mathcal{G})}{\sqrt{H(\mathcal{C})H(\mathcal{G})}}$$

Gaussian mixture models

The Gaussian Generative Model

- We'll be working with the following generative model for data $\mathcal{D}$
- Assume a datapoint $\boldsymbol{x}$ is generated as follows:
  - Choose a cluster z from $\{1, \ldots, K\}$ such that $\mathbb{P}(z = k) = \pi_k$
  - Given z, sample $\boldsymbol{x}$ from a Gaussian distribution $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_z, \Sigma_z)$

- Can also be written:

$$\mathbb{P}(z = k) = \pi_k$$

$$\mathbb{P}(\boldsymbol{x}|z = k) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \Sigma_k)$$

Clusters from Generative Model

- This defines joint distribution

$$p(z, \boldsymbol{x}) = p(z)p(\boldsymbol{x}|z)$$

with parameters $\{\pi_k, \boldsymbol{\mu}_k, \Sigma_k\}_{k=1}^K$

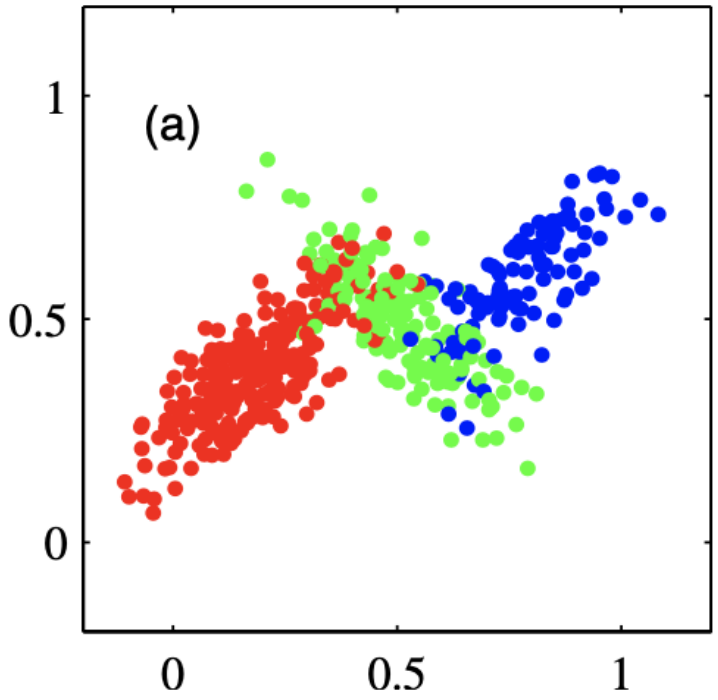
- The marginal of is given by $p(\boldsymbol{x}) = \sum_z p(z, \boldsymbol{x})$

- The a posteriori probability $p(z|\boldsymbol{x})$ can be computed using Bayes rule

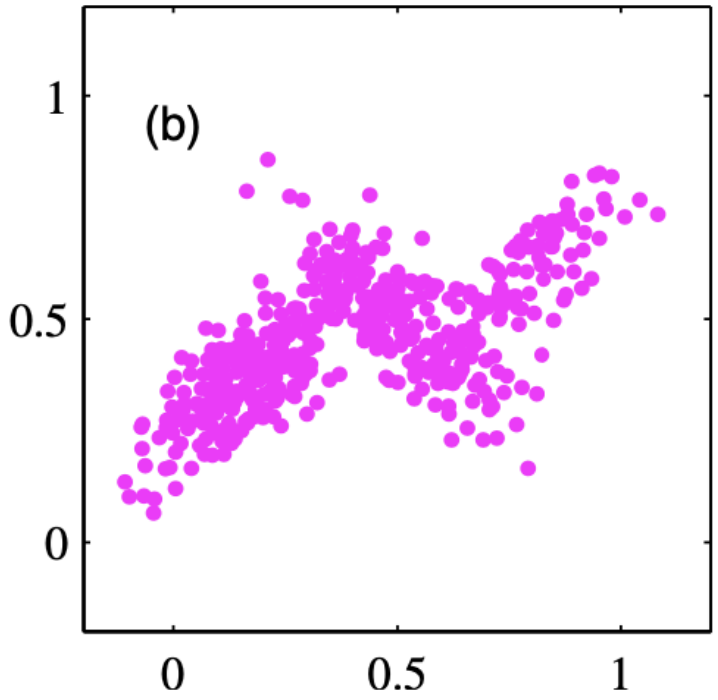
$$p(z = k|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|z = k)p(z = k)}{p(\boldsymbol{x})}$$

- This tells us the probability that $\boldsymbol{x}$ comes from the k-th cluster

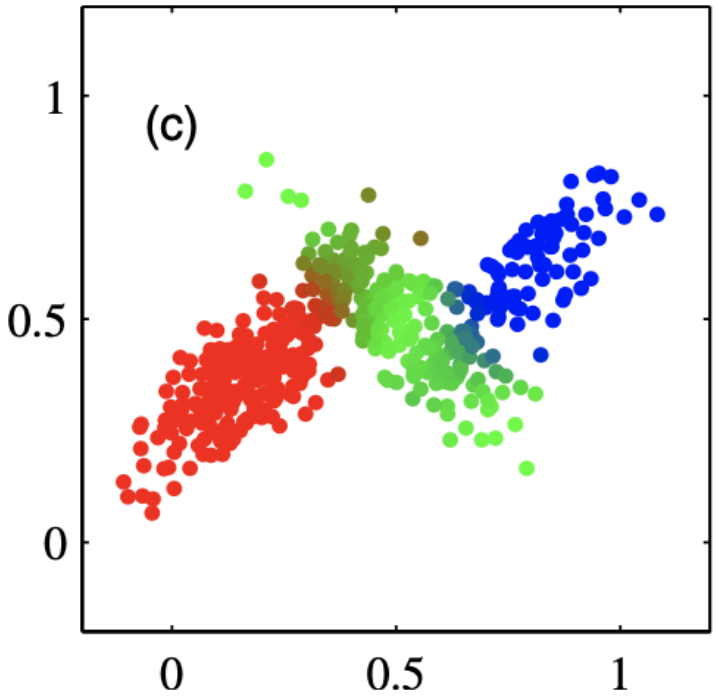
Illustration



$p(\boldsymbol{x}|z)$



$p(\boldsymbol{x})$



$p(z|\boldsymbol{x})$

Maximum Likelihood with Latent Variables

- How should we choose the parameters $\{\pi_k, \boldsymbol{\mu}_k, \Sigma_k\}_{k=1}^K$?
- Maximum likelihood principle: choose parameters to maximize likelihood of observed data
- We don't observe the cluster assignments z, we only see the data $\boldsymbol{x}$
- Given data $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$, choose parameters to maximize:

$$\log p(\mathcal{D}) = \sum_{i=1}^{N} \log p(\boldsymbol{x}_i)$$

- We can find $p(\boldsymbol{x})$ by marginalizing out z:

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \log p(z = k, \boldsymbol{x}) = \sum_{k=1}^{K} \log p(\boldsymbol{x}|z = k)p(z = k)$$

GMM

- What is $p(\boldsymbol{x})$?

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \log p(\boldsymbol{x}|z=k)p(z=k) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \Sigma_k)$$

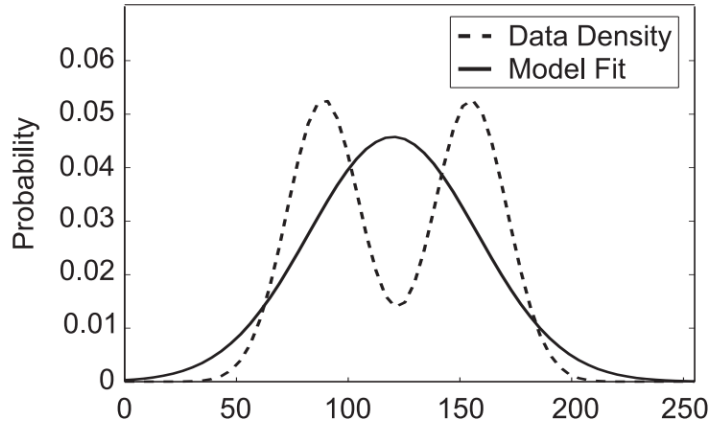
- This distribution is an example of a Gaussian Mixture Model (GMM), and π_k are known as the mixing coefficients

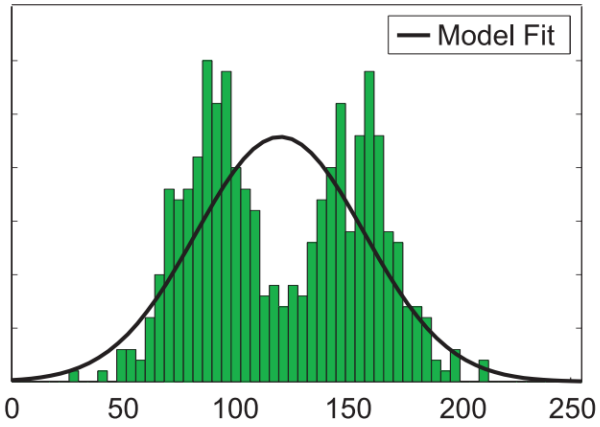
- In general, we would have different covariance for each cluster, i.e., $p(\boldsymbol{x}|z=k) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \Sigma_k)$.

- If we allow arbitrary covariance matrices, GMMs are universal approximators of densities (if you have enough Gaussians). Even diagonal GMMs are universal approximators.

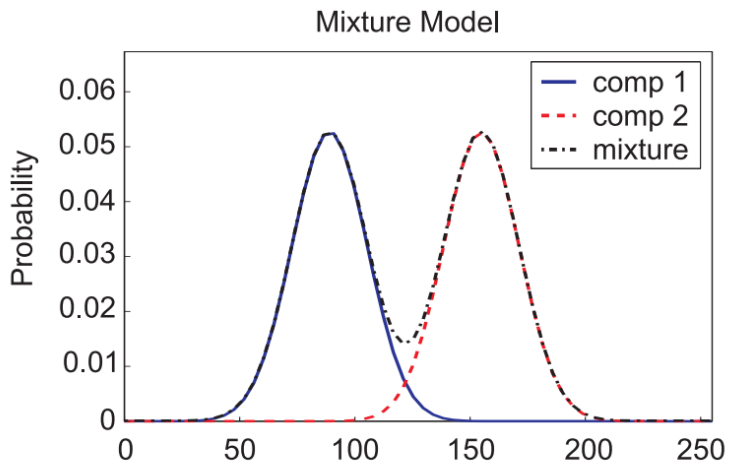
Visualizing a Mixture of Gaussians – 1D Gaussians

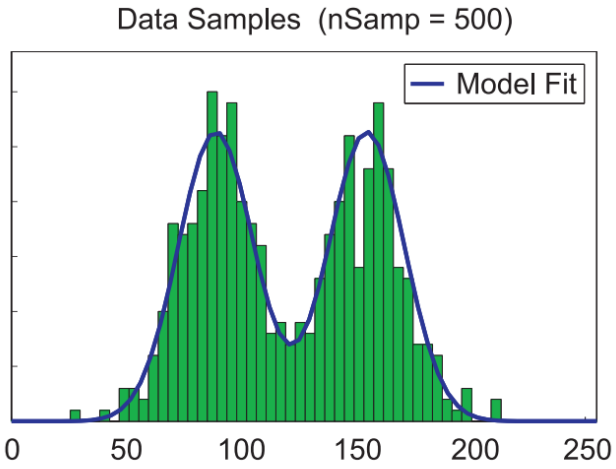
- If you fit one Gaussian distribution to data:



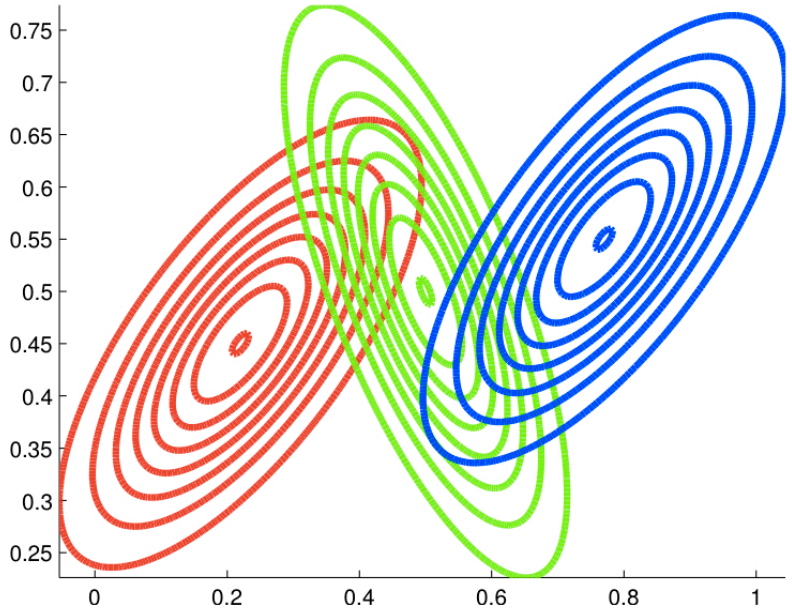


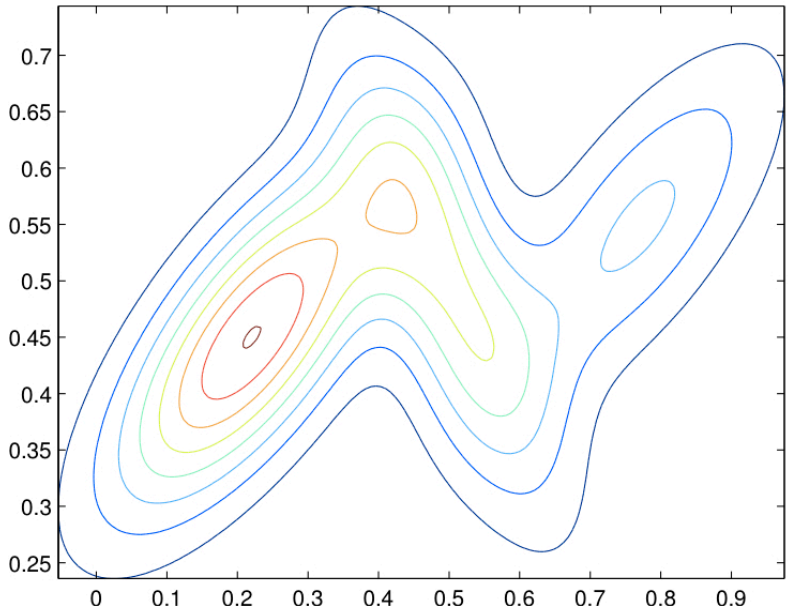
- Now, we are trying to fit a GMM with $K = 2$:

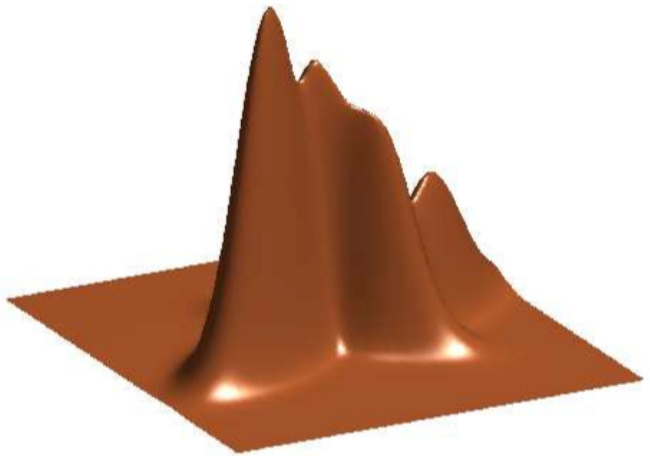




Visualizing a Mixture of Gaussians – 2D Gaussians







Fitting GMMs: Maximum Likelihood

- Maximum likelihood objective:

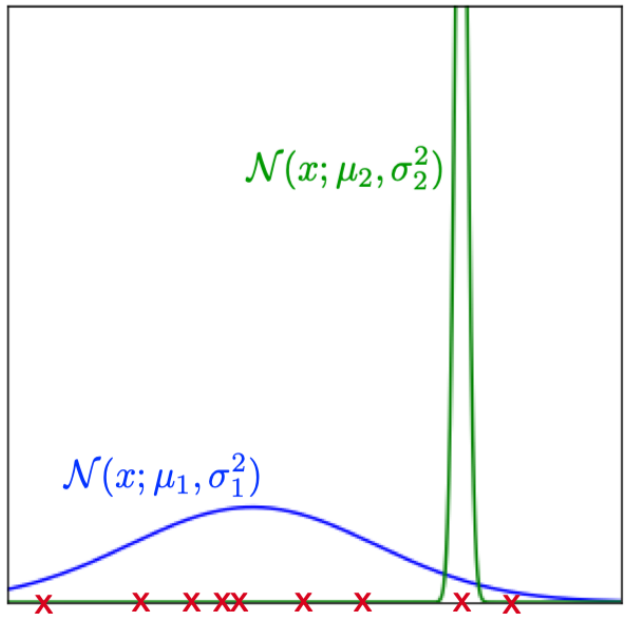
$$\log p(\mathcal{D}) = \sum_{i=1}^{N} \log p(\boldsymbol{x}_i) = \sum_{i=1}^{N} \log \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_i | \boldsymbol{\mu}_k, \Sigma_k) \right)$$

- In general, no closed-form solution
- Not identifiable: solution is invariant to permutations Challenges in optimizing this using gradient descent?
- Non-convex (due to permutation symmetry)
- Need to enforce non-negativity constraint on π_k and PSD constraint on Σ_k
- Derivatives w.r.t. Σ_k are expensive/complicated
- We need a different approach!

PSD: positive semidefinite

Fitting GMMs: Maximum Likelihood

- Warning: you don't want the global maximum. You can achieve arbitrarily high training likelihood by placing a small-variance Gaussian component on a training example.
- This is known as a singularity.



Maximum Likelihood

- For the derivation of the maximum likelihood, we assume $\Sigma_k = I$ for simplicity.
- Observation: if we knew z_i for every $\boldsymbol{x}_i$, i.e. our dataset was $\mathcal{D}_{\text{complete}} = \{(z_i, \boldsymbol{x}_i)\}_{i=1}^N$ the maximum likelihood problem is easy:

$$\log p\left(\mathcal{D}_{\text{complete}}\right) = \sum_{i=1}^{N} \log p(z_i, \boldsymbol{x}_i)$$

$$= \sum_{i=1}^{N} \log p(\boldsymbol{x}_i | z_i) + \log p(z_i\ )$$

$$= \sum_{i=1}^{N} \left(\sum_{k=1}^{K} \mathbb{I}\{z_i = k\} \left(\log \mathcal{N}(\boldsymbol{x}_i | \boldsymbol{\mu}_k, I) + \log \pi_k\right) \right)$$

Maximum Likelihood

$$\max_{\boldsymbol{\mu}_k, \pi_k} \log p\left(\mathcal{D}_{\text{complete}}\right) = \max_{\boldsymbol{\mu}_k, \pi_k} \sum_{i=1}^{N} \sum_{k=1}^{K} \mathbb{I}\{z_i = k\} \left(\log \mathcal{N}\left(\boldsymbol{x}_i | \boldsymbol{\mu}_k, I\right) + \log \pi_k\right)$$

- By maximizing $\log p\left(\mathcal{D}_{\text{complete}}\right)$, we would get this:

$$\widehat{\boldsymbol{\mu}}_k = \frac{\sum_{i=1}^{N} \mathbb{I}\{z_i = k\} \boldsymbol{x}_i}{\sum_{i=1}^{N} \mathbb{I}\{z_i = k\}}$$

$$\hat{\pi}_k = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\{z_i = k\}$$

Maximum Likelihood

- We haven't observed the cluster assignments z_i, but we can compute $p(z_i|\boldsymbol{x}_i)$ using Bayes rule

- Conditional probability (using Bayes rule) of z given $\boldsymbol{x}$

$$p(z = k|\boldsymbol{x}) = \frac{p(z = k)p(\boldsymbol{x}|z = k)}{p(\boldsymbol{x})}$$
$$= \frac{p(z = k)p(\boldsymbol{x}|z = k)}{\sum_{k=1}^{K} p(z = k)p(\boldsymbol{x}|z = k)}$$
$$= \frac{\pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, I)}{\sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, I)}$$

Maximum Likelihood

$$\log p\left(\mathcal{D}_{\text{complete}}\right) = \sum_{i=1}^{N}\sum_{k=1}^{K} \mathbb{I}\{z_i = k\}\left(\log \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_k, I) + \log \pi_k\right)$$

- We don't know the cluster assignments $\mathbb{I}\{z_i = k\}$ (they are our latent variables), but we know their expectation $\mathbb{E}(\mathbb{I}\{z_i = k\}|\boldsymbol{x}_i) = p(z_i = k|\boldsymbol{x}_i)$ when $\boldsymbol{\mu}_k$, π_k are known (see the previous slide)

- If we plug in $r_{i,k} = p(z_i = k|\boldsymbol{x}_i)$ for $\mathbb{I}\{z_i = k\}$, we get:

$$\sum_{i=1}^{N}\sum_{k=1}^{K} r_{i,k}\left(\log \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_k, \Sigma_k) + \log \pi_k\right)$$

- This is still easy to optimize! Solution is similar to what we have seen:

$$\hat{\boldsymbol{\mu}}_k = \frac{\sum_{i=1}^{N} r_{i,k}\boldsymbol{x}_i}{\sum_{i=1}^{N} r_{i,k}}$$

$$\hat{\pi}_k = \frac{1}{N}\sum_{i=1}^{N} r_{i,k}$$

Note: this only works if we treat $r_{i,k}$ as constant.

How Can We Fit a Mixture of Gaussians?

- This motivates the Expectation-Maximization algorithm, which alternates between two steps:

1. **E-step:** Compute the posterior probabilities $r_{i,k} = p(z_i = k|\boldsymbol{x}_i)$ given our current model $(\boldsymbol{\mu}_k, \pi_k)$, i.e., how much do we think a cluster is responsible for generating a datapoint.

2. **M-step:** Use the equations $(\hat{\boldsymbol{\mu}}_k, \hat{\pi}_k)$ on the last slide to update the parameters, assuming $r_{i,k}$ are held fixed, i.e., change the parameters of each Gaussian to maximize the probability that it would generate the data it is currently responsible for.

EM Algorithm for GMM

- Initialize the means $\hat{\boldsymbol{\mu}}_k$ and mixing coefficients $\hat{\pi}_k$
- Iterate until convergence:
  - E-step: Evaluate the responsibilities $r_{i,k}$ given current parameters

$$r_{i,k} = p(z_i = k|\boldsymbol{x}_i) = \frac{\hat{\pi}_k \mathcal{N}(\boldsymbol{x}_i|\hat{\boldsymbol{\mu}}_k, I)}{\sum_{j=1}^{K} \hat{\pi}_j \mathcal{N}(\boldsymbol{x}_i|\hat{\boldsymbol{\mu}}_j, I)} = \frac{\hat{\pi}_k \exp\left\{-\frac{1}{2}\|\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k\|^2\right\}}{\sum_{j=1}^{K} \hat{\pi}_j \exp\left\{-\frac{1}{2}\|\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_j\|^2\right\}}$$

  - M-step: Re-estimate the parameters given current responsibilities

$$\hat{\boldsymbol{\mu}}_k = \frac{1}{\sum_{i=1}^{N} r_{i,k}} \sum_{i=1}^{N} r_{i,k} \boldsymbol{x}_i$$

$$\hat{\pi}_k = \frac{\sum_{i=1}^{N} r_{i,k}}{N}$$

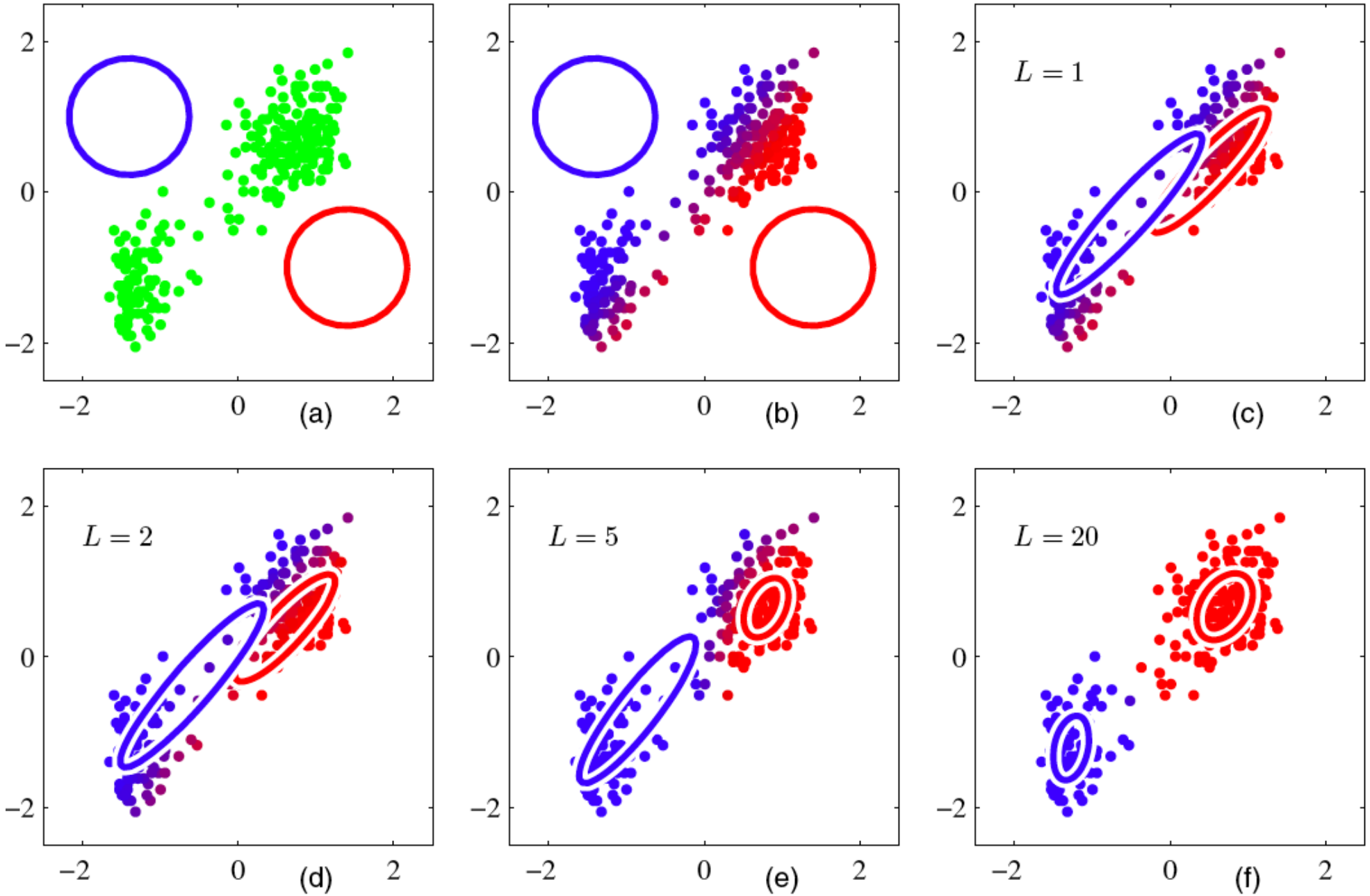
- Evaluate log likelihood and check for convergence

$$\log p(\mathcal{D}) = \sum_{i=1}^{N} \log \left(\sum_{k=1}^{K} \hat{\pi}_k \mathcal{N}(\boldsymbol{x}_i|\hat{\boldsymbol{\mu}}_k, I) \right)$$

Parameter initialization

Followed by cluster initialization

First iteration $L = 1$



Relation to k-Means

- The K-Means Algorithm:
  1. **Assignment step:** Assign each data point to the closest cluster
  2. **Refitting step:** Move each cluster centre to the average of the data assigned to it
- The EM Algorithm:
  1. **E-step:** Compute the posterior probability over z given our current model
  2. **M-step:** Maximize the probability that it would generate the data it is currently responsible for.
- Can you find the similarities between the soft k-Means algorithm and EM algorithm with shared covariance $\sigma^2 I$?
- Both rely on alternating optimization methods and can suffer from bad local optima.

Decomposing signals in components (matrix factorization problems)

Main methods

- **Principal component analysis (PCA)**
- **Kernel Principal Component Analysis (kPCA)**
- Truncated singular value decomposition and latent semantic analysis
- Dictionary Learning
- Factor Analysis
- Independent component analysis (ICA)
- Non-negative matrix factorization (NMF or NNMF)
- Latent Dirichlet Allocation (LDA)

PCA

Principal Components Analysis

- **Idea:**
  - Given data points in a d-dimensional space, project into lower dimensional space while preserving as much information as possible
    - Eg, find best planar approximation to 3D data
    - Eg, find best 12-D approximation to 10^4-D data
  - In particular, choose projection that minimizes squared error in reconstructing original data

PCA algorithm

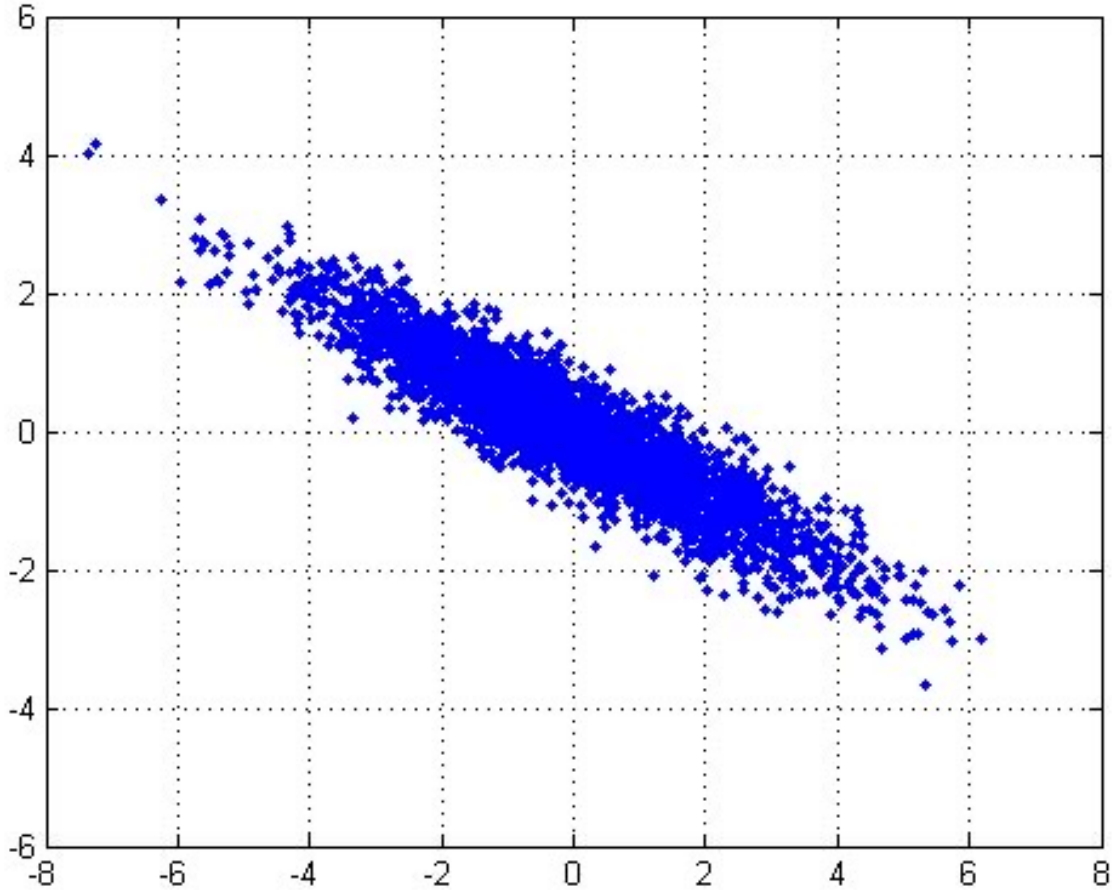
- Given data $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$, compute the covariance matrix estimate $\hat{\Sigma}$

$$\hat{\Sigma} = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \overline{\boldsymbol{x}})(\boldsymbol{x}_i - \overline{\boldsymbol{x}})^T$$

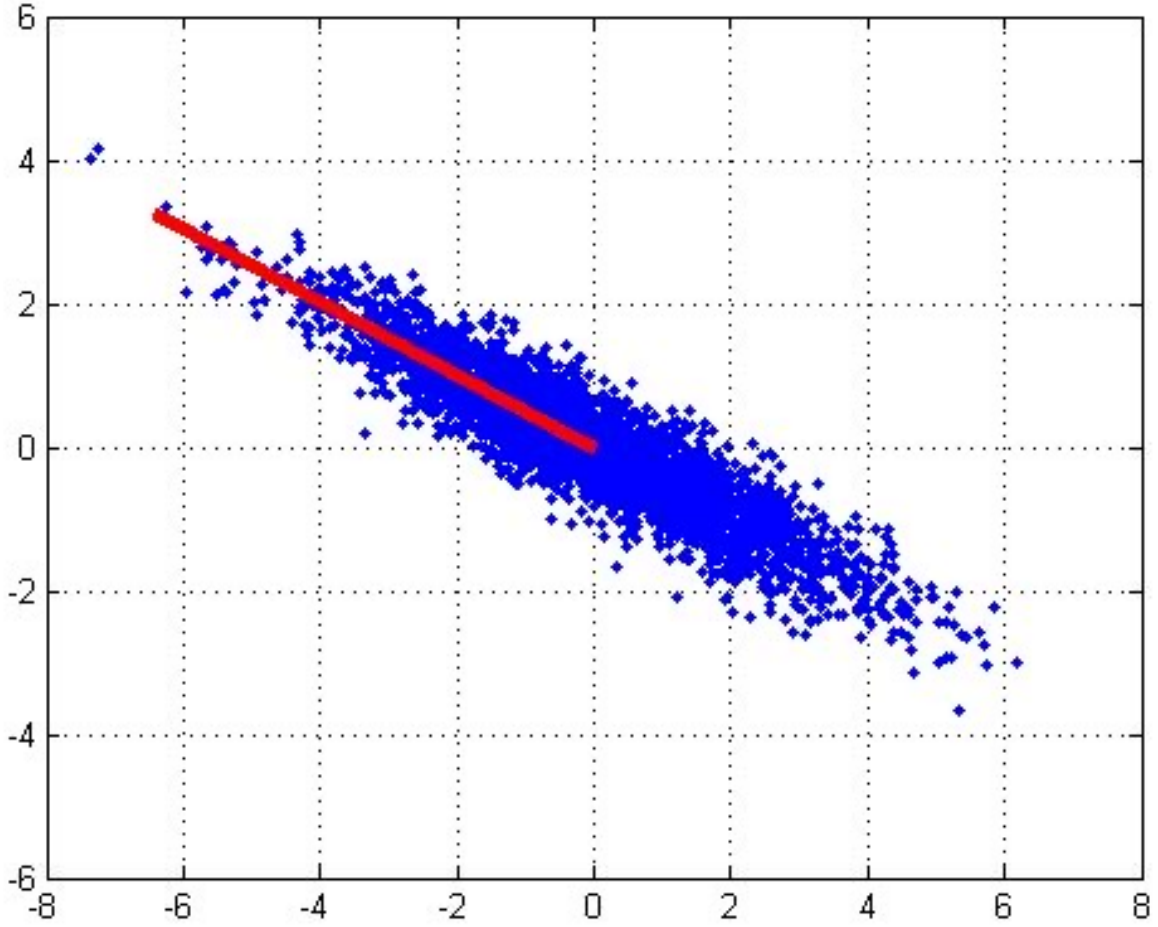
$$\overline{\boldsymbol{x}} = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{x}_i$$

- PCA basis vectors = the eigenvectors of $\hat{\Sigma}$
  - $\{\lambda_i, \boldsymbol{u}_i\}_{i=1,\ldots,N}$ eigenvectors/eigenvalues of $\hat{\Sigma}$ with $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N$
  - Larger eigenvalue $\Rightarrow$ more important eigenvectors
  - Best approximation of dimension k with the top k principal components

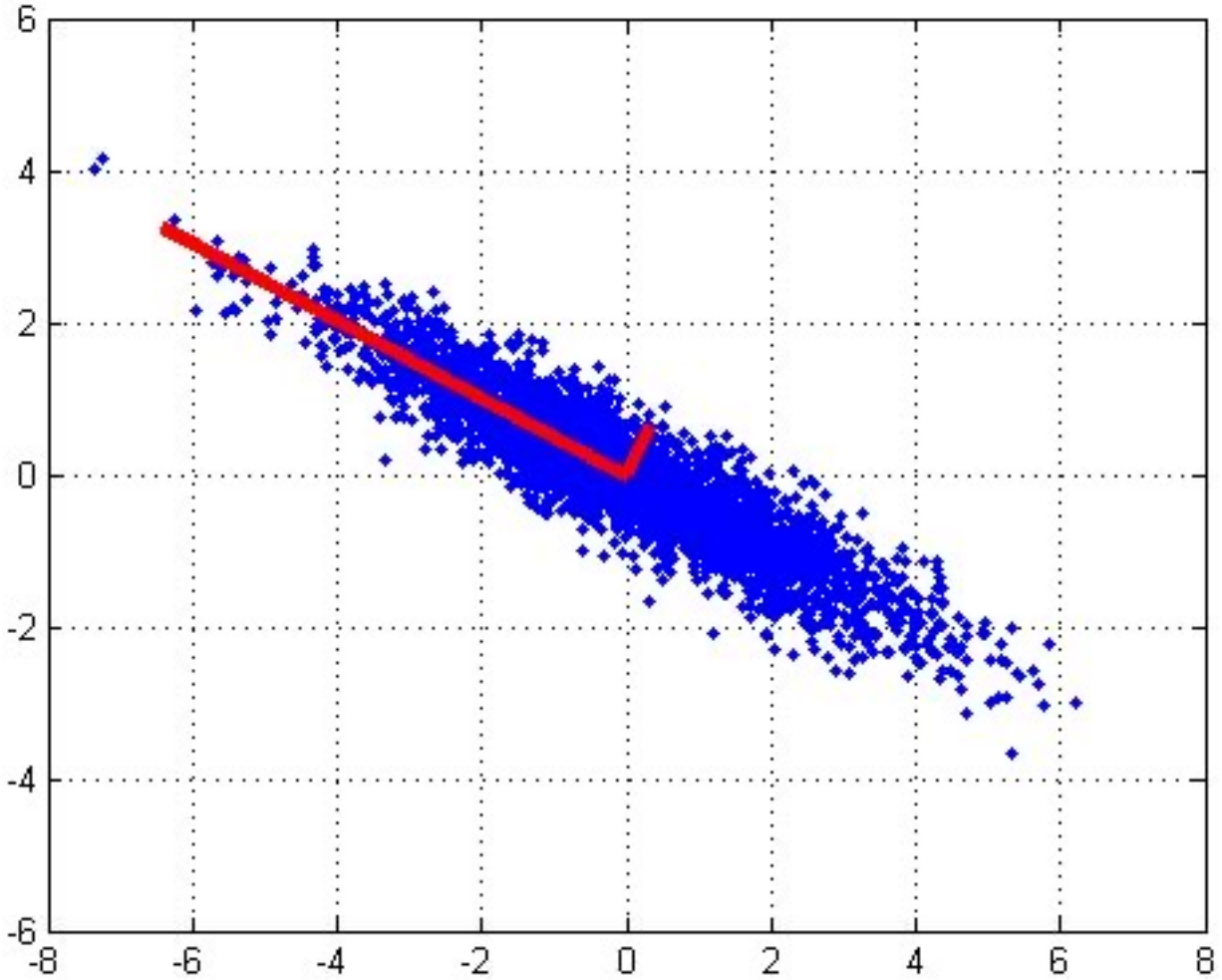
2D Gaussian dataset



1st PCA axis

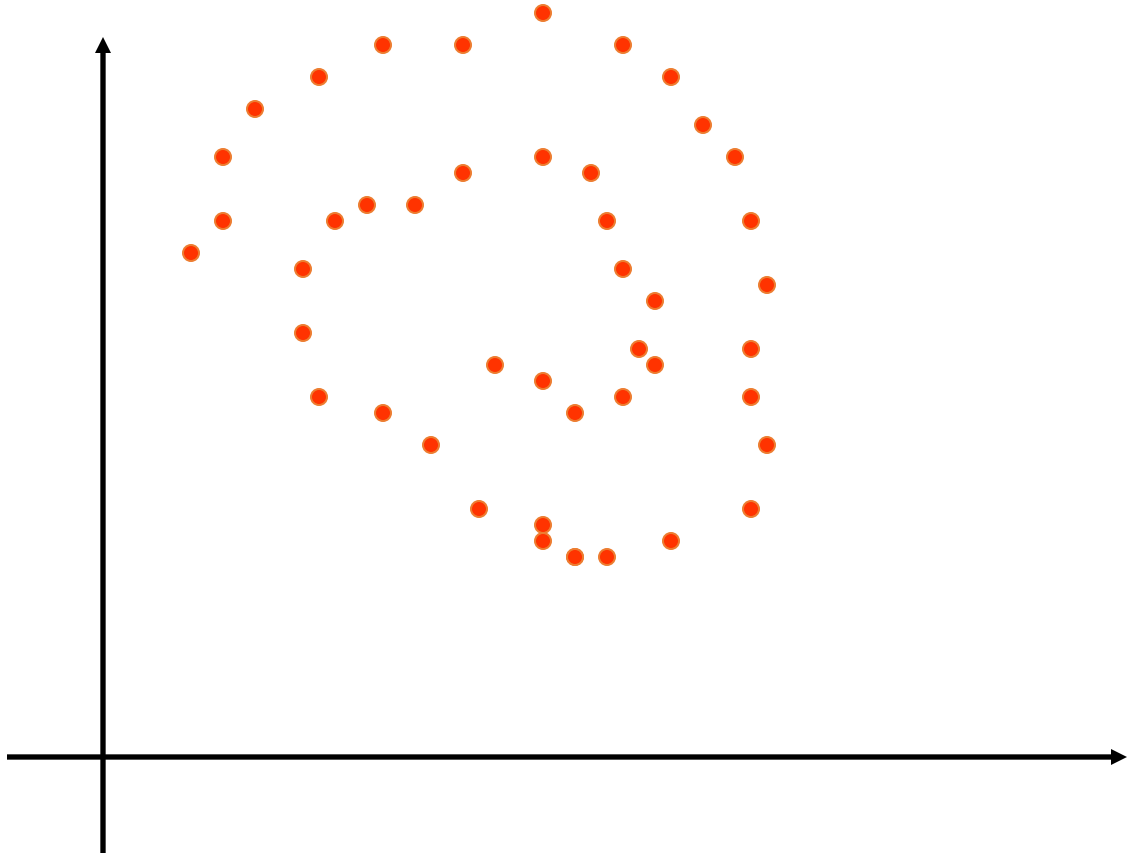


2nd PCA axis



Kernel PCA

PCA, a Problematic Data Set

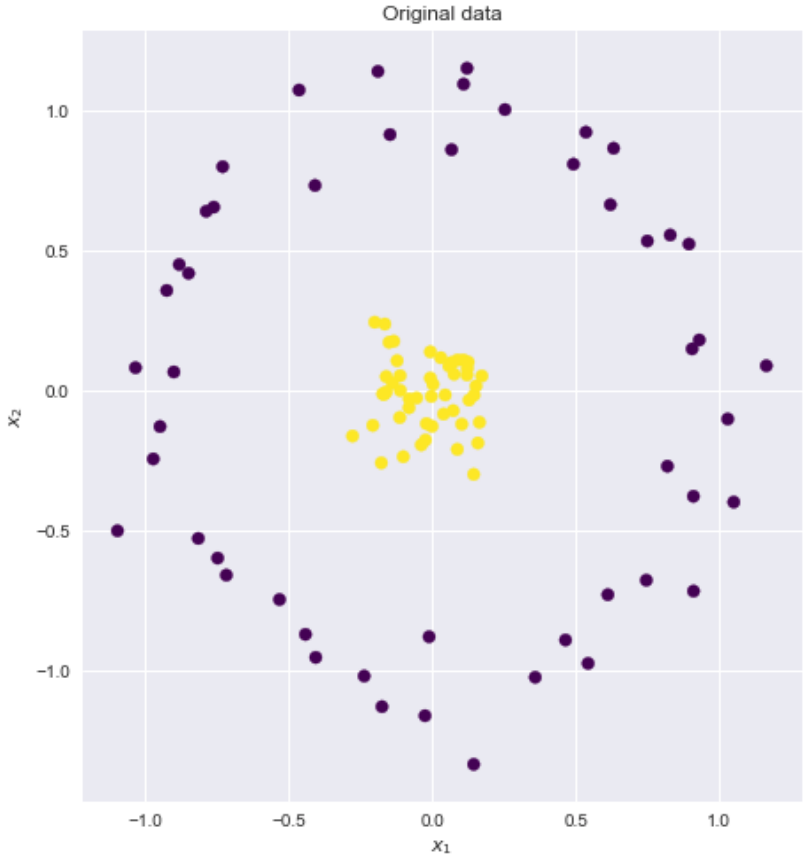


PCA cannot capture NON-LINEAR structure!

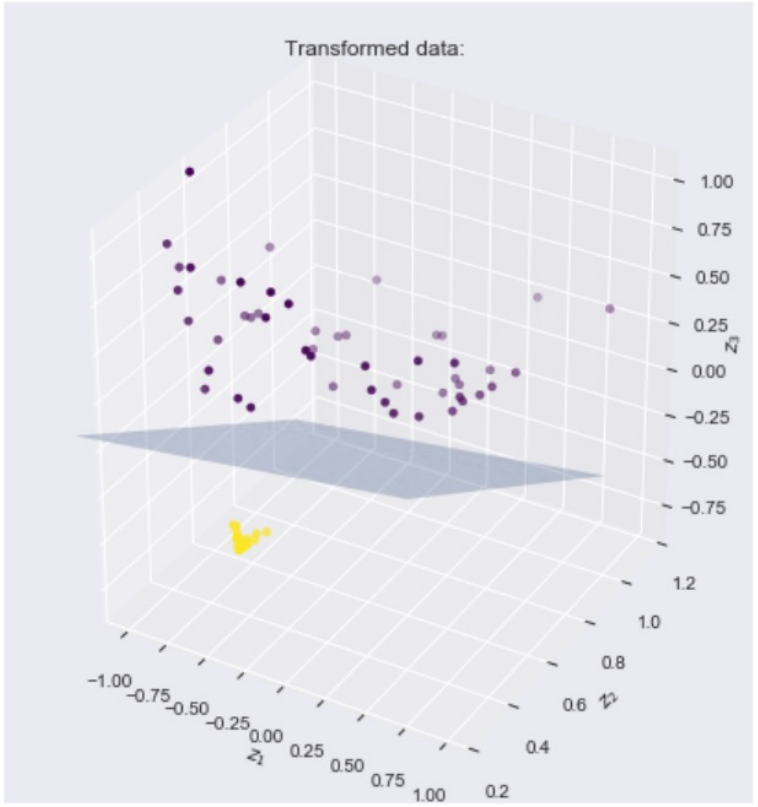
Making PCA non-linear

- Suppose that instead of using the points $\boldsymbol{x}_i \in \mathbb{R}^p$ as is, we wanted to go to some different feature space $\phi(\boldsymbol{x}_i) \in \mathbb{R}^q$
  - E.g. using polar coordinates instead of cartesian coordinates would help us deal with the circle
- In the higher dimensional space, we can then do PCA
- The result will be non-linear in the original data space!

Illustration of feature mapping ϕ



$$\phi(x_1, x_2) = \begin{pmatrix} x_1 x_2 \sqrt{2} \\ x_1^2 \\ x_2^2 \end{pmatrix}$$



PCA in feature space

- Suppose for the moment that the mean of the data in feature space is 0, so:

$$\frac{1}{N}\sum_{i=1}^{N}\phi(\boldsymbol{x}_i) = 0$$

- The covariance matrix is:

$$C = \frac{1}{N}\sum_{i=1}^{N}\phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_i)^T$$

- The eigenvectors are

$$C\boldsymbol{v}_j = \lambda_j \boldsymbol{v}_j, \qquad j = 1, \ldots, q$$

- We want to avoid explicitly going to feature space - instead we want to work with kernels

$$K(\boldsymbol{x}_i, \boldsymbol{x}_k) = \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_k)$$

PCA in feature space

- Rewrite the PCA equation:

$$C\boldsymbol{v}_j = \frac{1}{N}\sum_{i=1}^{N}\phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_i)^T\boldsymbol{v}_j = \lambda_j\boldsymbol{v}_j, \qquad j = 1, \dots, q$$

$$\frac{1}{N\lambda_j}\sum_{i=1}^{N}\left(\boldsymbol{v}_j{}^T\phi(\boldsymbol{x}_i)\right)\phi(\boldsymbol{x}_i) = \boldsymbol{v}_j$$

- So the eigenvectors can be written as linear combination for features

$$\boldsymbol{v}_j = \sum_{i=1}^{N} a_{ji}\phi(\boldsymbol{x}_i)$$

with $a_{ji} = \frac{1}{N\lambda_j}\boldsymbol{v}_j{}^T\phi(\boldsymbol{x}_i) \in \mathbb{R}$

- Finding the eigenvectors is equivalent to finding the coefficients $a_{ij}, j = 1, \dots, q, i = 1, \dots, N$

PCA in feature space

- By substituting this back into the equation $C\boldsymbol{v}_j = \lambda_j \boldsymbol{v}_j$, we get:

$$\frac{1}{N}\sum_{i=1}^{N} \phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_i)^T \left(\sum_{\ell=1}^{N} a_{j\ell}\phi(\boldsymbol{x}_\ell)\right) = \lambda_j \sum_{\ell=1}^{N} a_{j\ell}\phi(\boldsymbol{x}_\ell)\,, \qquad j = 1, \ldots, q$$

- We can rewrite this as

$$\frac{1}{N}\sum_{i=1}^{N} \phi(\boldsymbol{x}_i)\left(\sum_{\ell=1}^{N} a_{j\ell}\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_\ell)\right) = \frac{1}{N}\sum_{i=1}^{N} \phi(\boldsymbol{x}_i)\left(\sum_{\ell=1}^{N} a_{j\ell}\ K(\boldsymbol{x}_i, \boldsymbol{x}_\ell)\right) = \lambda_j \sum_{\ell=1}^{N} a_{j\ell}\phi(\boldsymbol{x}_\ell)$$

- Multiply this by $\phi(\boldsymbol{x}_k)^T$ to the left:

$$\frac{1}{N}\sum_{i=1}^{N} \phi(\boldsymbol{x}_k)^T\phi(\boldsymbol{x}_i)\left(\sum_{\ell=1}^{N} a_{j\ell}\ K(\boldsymbol{x}_i, \boldsymbol{x}_\ell)\right) = \lambda_j \sum_{\ell=1}^{N} a_{j\ell}\phi(\boldsymbol{x}_k)^T\phi(\boldsymbol{x}_\ell)$$

PCA in feature space

- This yields

$$\frac{1}{N}\sum_{i=1}^{N} K(\boldsymbol{x}_k, \boldsymbol{x}_i)\left(\sum_{\ell=1}^{N} a_{j\ell}\ K(\boldsymbol{x}_i, \boldsymbol{x}_\ell)\right) = \lambda_j \sum_{\ell=1}^{N} a_{j\ell} K(\boldsymbol{x}_k, \boldsymbol{x}_\ell)$$

$$\sum_{\ell=1}^{N}\left(\sum_{i=1}^{N} K(\boldsymbol{x}_k, \boldsymbol{x}_i) K(\boldsymbol{x}_i, \boldsymbol{x}_\ell)\right) a_{j\ell} = N\lambda_j \sum_{\ell=1}^{N} K(\boldsymbol{x}_k, \boldsymbol{x}_\ell) a_{j\ell}$$

- Equivalent to the matrix form

$$K^2 \boldsymbol{a}_j = N\lambda_j K \boldsymbol{a}_j$$

where $K = \left[K(\boldsymbol{x}_i, \boldsymbol{x}_j)\right]_{i,j=1,\ldots,N}$ and $\boldsymbol{a}_j = \left[a_{j\ell}\right]_{\ell=1,\ldots,N}$

- We can remove a factor of K from both sides of the equation

$$K \boldsymbol{a}_j = N\lambda_j \boldsymbol{a}_j$$

  - Trivial if K is invertible,
  - Only affect eigenvectors with eigenvalues 0 if K is not invertible, which will not be principal components anyway.

PCA in feature space

- We have a normalization condition for the $\boldsymbol{a}_j$ vector:

$$\boldsymbol{v}_j{}^T\boldsymbol{v}_j = 1 \Rightarrow \left(\sum_{\ell=1}^{N} a_{j\ell}\phi(\boldsymbol{x}_\ell)\right)^T \sum_{k=1}^{N} a_{jk}\phi(\boldsymbol{x}_k) = \sum_{\ell=1}^{N}\sum_{k=1}^{N} a_{j\ell}a_{jk}\phi(\boldsymbol{x}_\ell)^T\phi(\boldsymbol{x}_k) = 1 \Rightarrow \boldsymbol{a}_j^T K\boldsymbol{a}_j = 1$$

- From $K\boldsymbol{a}_j = N\lambda_j\boldsymbol{a}_j$, it follows that

$$N\lambda_j\boldsymbol{a}_j^T\boldsymbol{a}_j = 1 \Rightarrow \left\|\boldsymbol{a}_j\right\|_2^2 = \frac{1}{N\lambda_j}$$

- For a new point $\boldsymbol{x}$, its projection onto the principal components $\boldsymbol{v}_j$ is:

$$\phi(\boldsymbol{x})^T\boldsymbol{v}_j = \sum_{i=1}^{N} a_{ji}\phi(\boldsymbol{x})^T\phi(\boldsymbol{x}_i) = \sum_{i=1}^{N} a_{ji}K(\boldsymbol{x},\boldsymbol{x}_i)$$

Normalizing the feature space

- In general, the features $\phi(\boldsymbol{x}_i)$ may not have mean 0
- We want to work with

$$\tilde{\phi}(\boldsymbol{x}_i) = \phi(\boldsymbol{x}_i) - \frac{1}{N}\sum_{k=1}^{N}\phi(\boldsymbol{x}_k)$$

- The corresponding kernel matrix entries are given by

$$\widetilde{K}(\boldsymbol{x}_i, \boldsymbol{x}_k) = \tilde{\phi}(\boldsymbol{x}_i)^T\tilde{\phi}(\boldsymbol{x}_k)$$

- After some algebra, we get

$$\widetilde{K} = K - \frac{2}{N}\mathbf{1}K + \frac{1}{N^2}\mathbf{1}K\mathbf{1}$$

where $\mathbf{1}$ is the matrix with all elements equal to $\mathbf{1}$

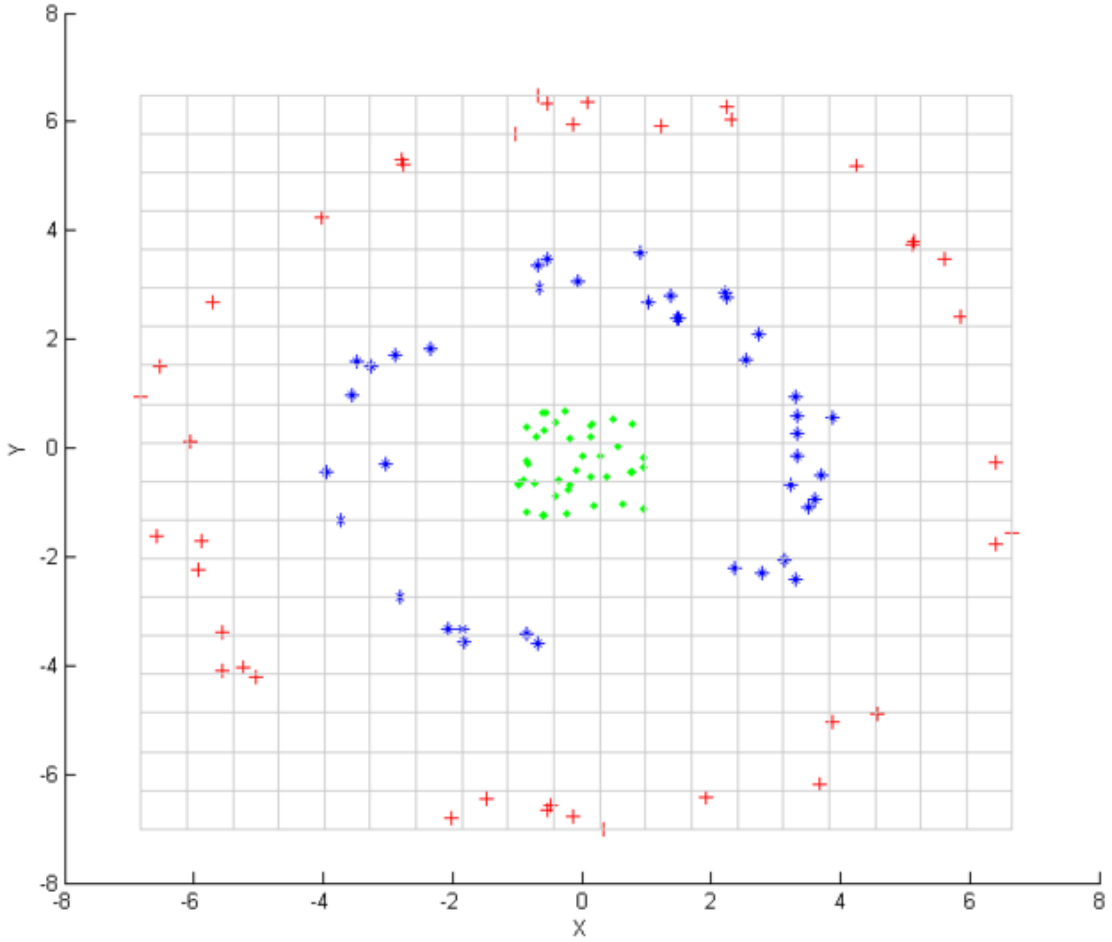
Summary of kernel PCA

1. Pick a kernel $K(\boldsymbol{x}, \boldsymbol{y})$
2. Construct the normalized kernel matrix $\widetilde{K}$ of the data (this will be of dimension $N \times N$)
3. Find the eigenvalues and eigenvectors of this matrix λ_j, a_{ji}
4. For any data point (new or old), we can represent it as the following set of features:

$$y_j = \sum_{i=1}^{N} a_{ji} \widetilde{K}(\boldsymbol{x}, \boldsymbol{x}_i) \, , j = 1, \dots, q$$

1. We can limit the number of components to $k < N$ for a more compact representation (by picking the $\boldsymbol{a}_j$'s corresponding to the highest eigenvalues)

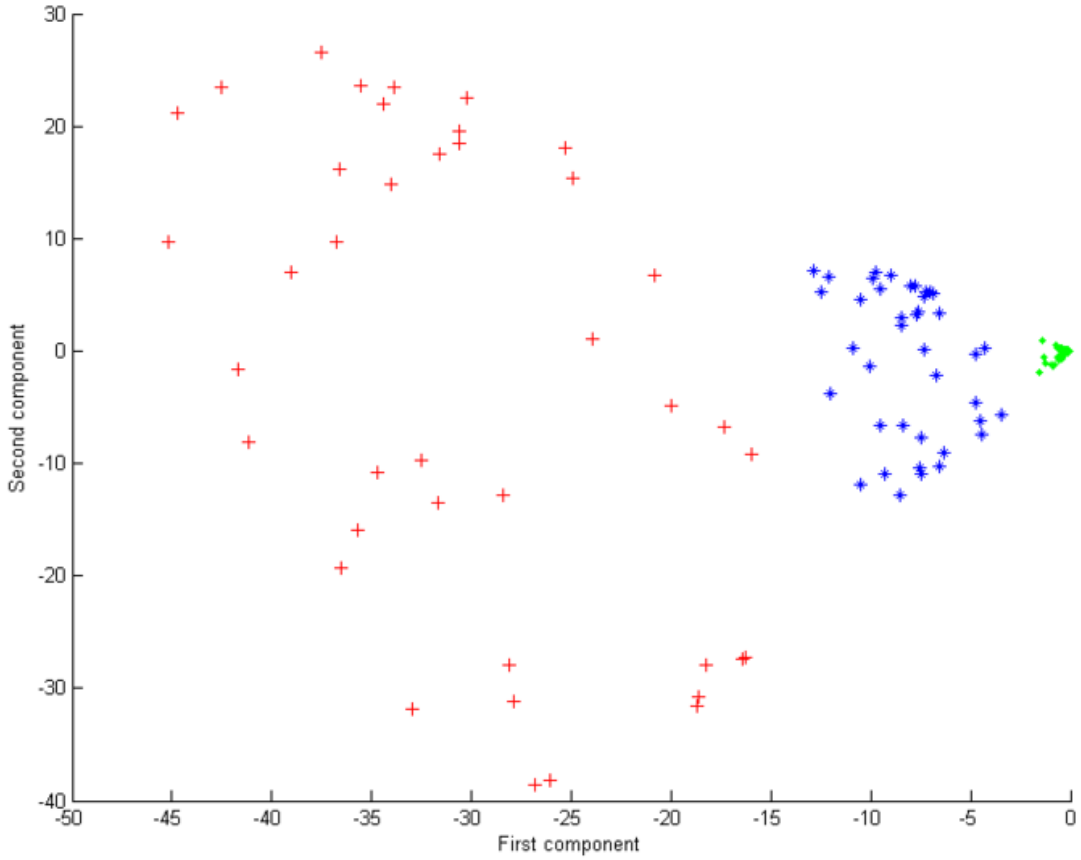
Input points before kernel PCA



http://en.wikipedia.org/wiki/Kernel_principal_component_analysis

Output after kernel PCA

- The three groups are distinguishable using the first component only



Kernel PCA

- **Goodness**
  - Can find non-linear principal components by projecting data into high dimensional space.
  - Do not need to define $\phi(\boldsymbol{x})$, instead define the kernel function $K(\boldsymbol{x}, \boldsymbol{y}) = \phi(\boldsymbol{x})^T \phi(\boldsymbol{y})$, and therefore it is possible to obtain very complex projection.
  - It is possible to obtain principal components even the coordinates of the data are unknown, but just the kernel value $K(\boldsymbol{x}, \boldsymbol{y})$.

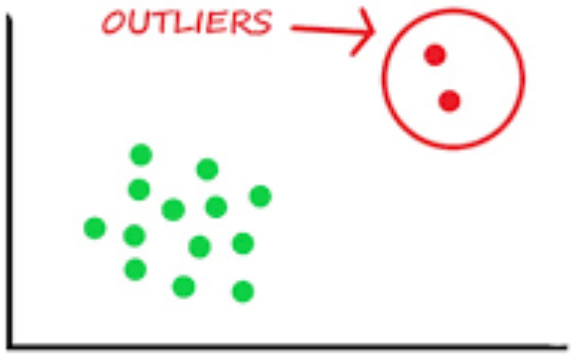
- **Problems**
  - Computing load is large, requires some approximation approach.
  - Difficult to find the exact approximation of $\boldsymbol{x}$ with the projections on the eigenspace.

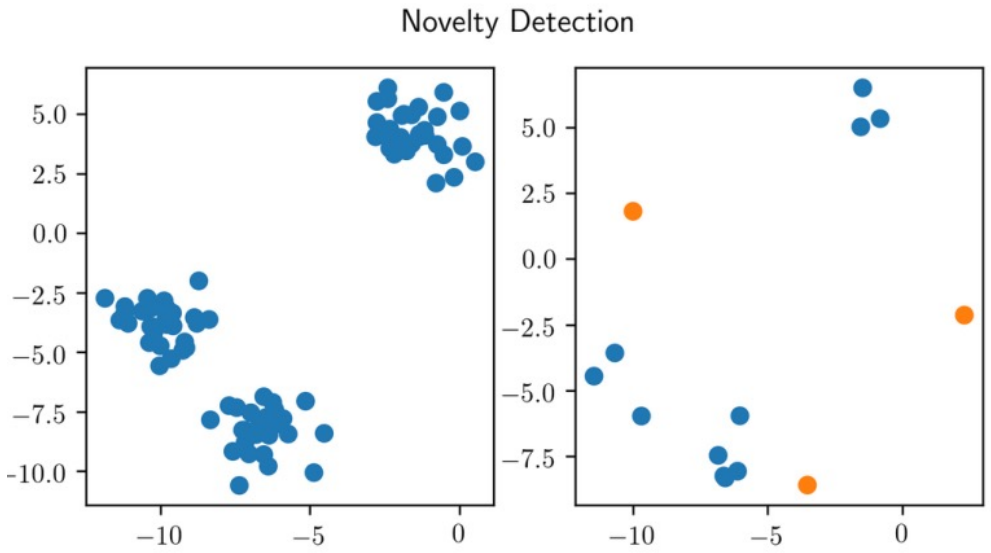
Novelty and outlier detection

Novelty and outlier detection

- The idea of novelty detection is to detect rare events, i.e. events that happen rarely, and hence, of which you have very little samples.
- The usual way of training a classifier will not work.
- Its usually used as a method for novelty detection which is also a kind of outlier detection problem. So how do you decide what a novel pattern is?
- Many approaches are based on the estimation of the density of probability for the data. Novelty corresponds to those samples where the density of probability is “very low”. How low depends on the application.

Some examples





https://amueller.github.io/aml/03-unsupervised-learning/03-outlier-detection.html

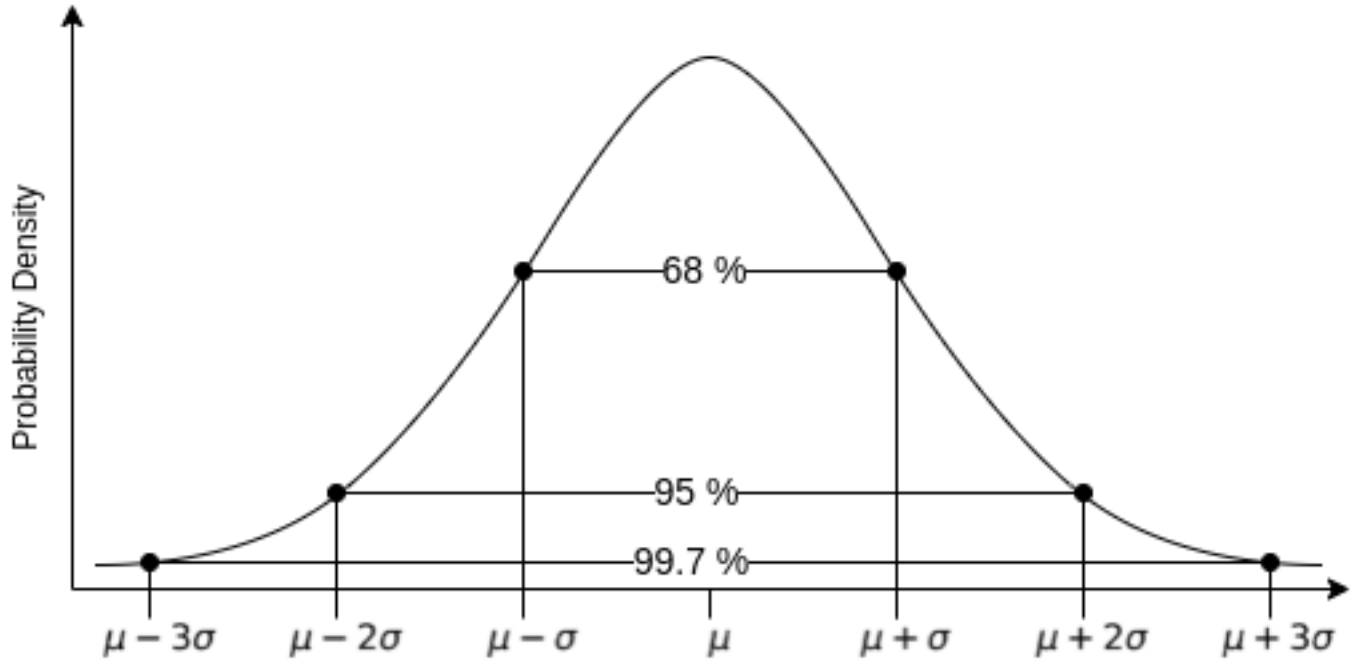
Z-score

- If the random variable x follows a normal probability distribution, then it will shift mean μ of sample to 0 and standard deviation σ to 1 using the formula

$$z = \frac{x - \mu}{\sigma}$$

- Other than feature scaling, this z-sore also serves to detect points that are possible outliers.
- Since we know that dataset following a gaussian distribution has property of 6-sigma rule
  - 68% data lies in first SD (Standard Deviation) from mean ($\mu \pm \sigma$)
  - 95% data lies in two SDs from mean ($\mu \pm 2\sigma$)
  - 99.7% data lies in three SDs from mean ($\mu \pm 3\sigma$)
- So the data points for which z-score is greater than 3 are going to serve as possible outlier points since they are quiet away from the mean of the sample.

Illustration of Z-score



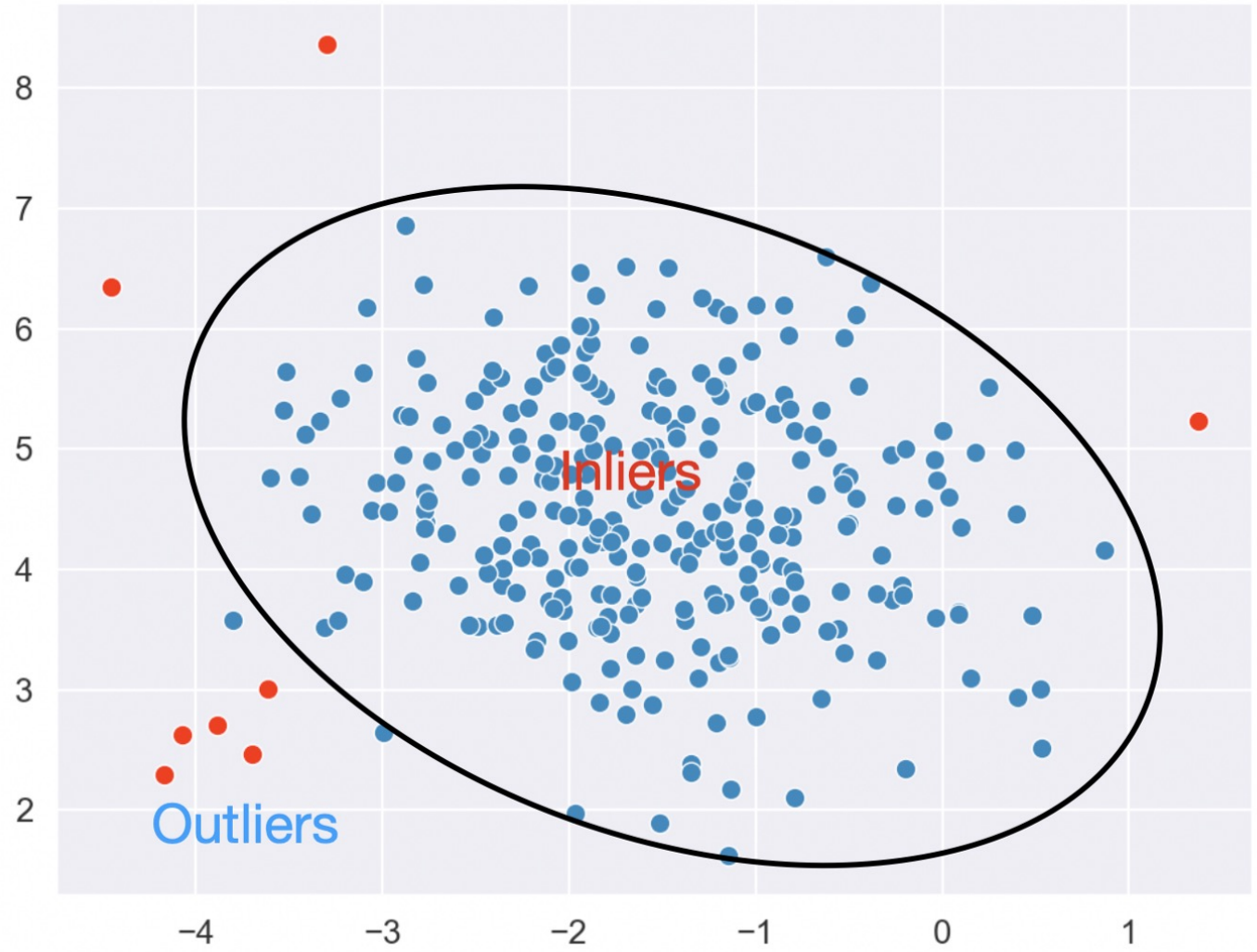
Elliptic Envelope or Robust Covariance

- It is the generalization of the z-score for a point $\boldsymbol{x}$ in the case of a p-dimensional multivariate probability distribution with some mean μ and covariance matrix Σ
- It is based on the Mahalanobis distance d, which is given by:

$$d(\boldsymbol{x}) = \sqrt{(\boldsymbol{x} - \mu)^T \Sigma^{-1} (\boldsymbol{x} - \mu)}$$

- Just like Z-score works well for assumption of normally distributed data, elliptic envelope tried to create an envelope around the normal distributed data and points outside the envelope are considered as outliers.
- The difficulties lie in the estimation of Σ that must be robust to outliers. Generally, it is desirable to estimate Σ without the the observations $\boldsymbol{x}_i$ with the largest distances $d(\boldsymbol{x}_i)$to the mean.

Illustration of elliptic envelope



One-class SVM

- SVMs are max-margin methods, i.e. they do not model a probability distribution. Here the idea is to find a function that is positive for regions with high density of points, and negative for small densities.

- One-Class SVM instead of using a hyperplane to separate two classes of instances, it uses a hypersphere to encompass all of the instances.

- Think of the “margin” as referring to the outside of the hypersphere, so by “the largest possible margin”, we mean “the smallest possible hypersphere”.

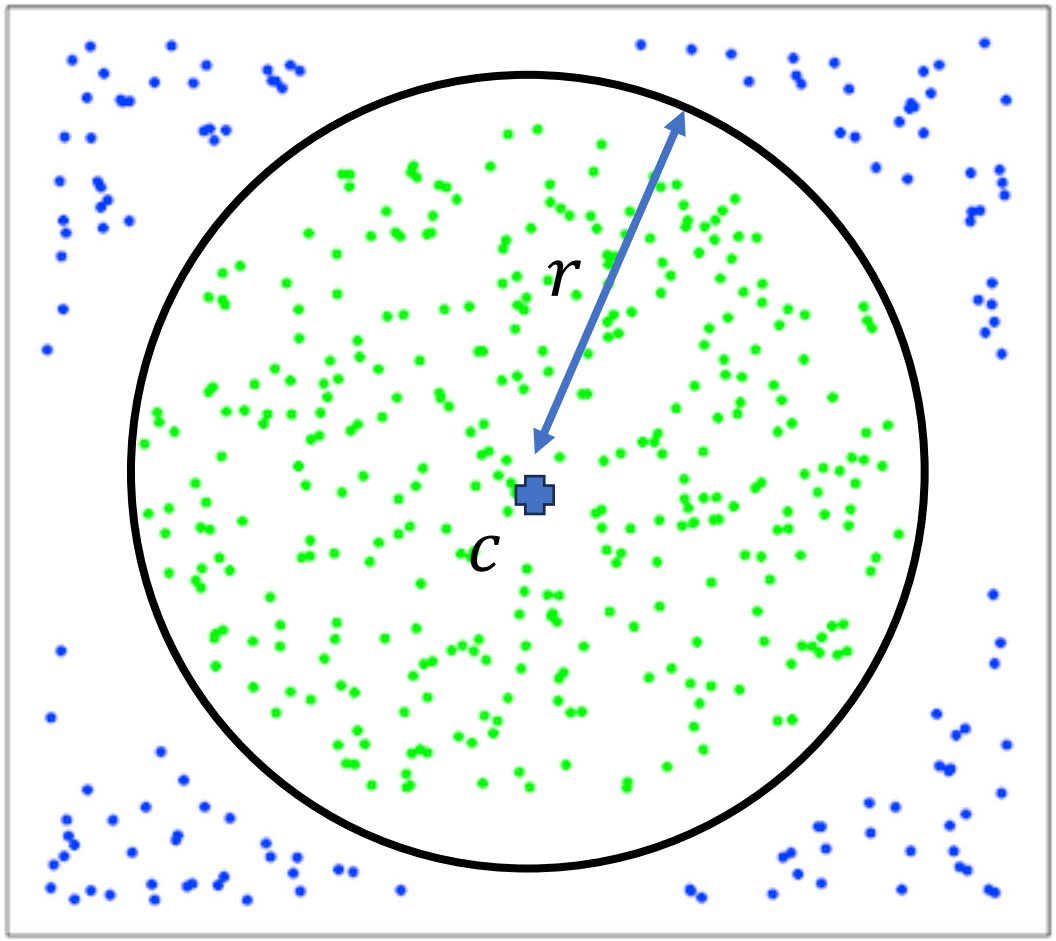
One-class SVM

- SVM based one-class classification (OCC) relies on identifying the smallest hypersphere (with radius r, and center c) consisting of all the data points.

$$\min_{r,c} r^2$$

$$\|\boldsymbol{x}_i - c\|^2 \leq r^2, \forall i = 1, \ldots, N$$

for a dataset $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$

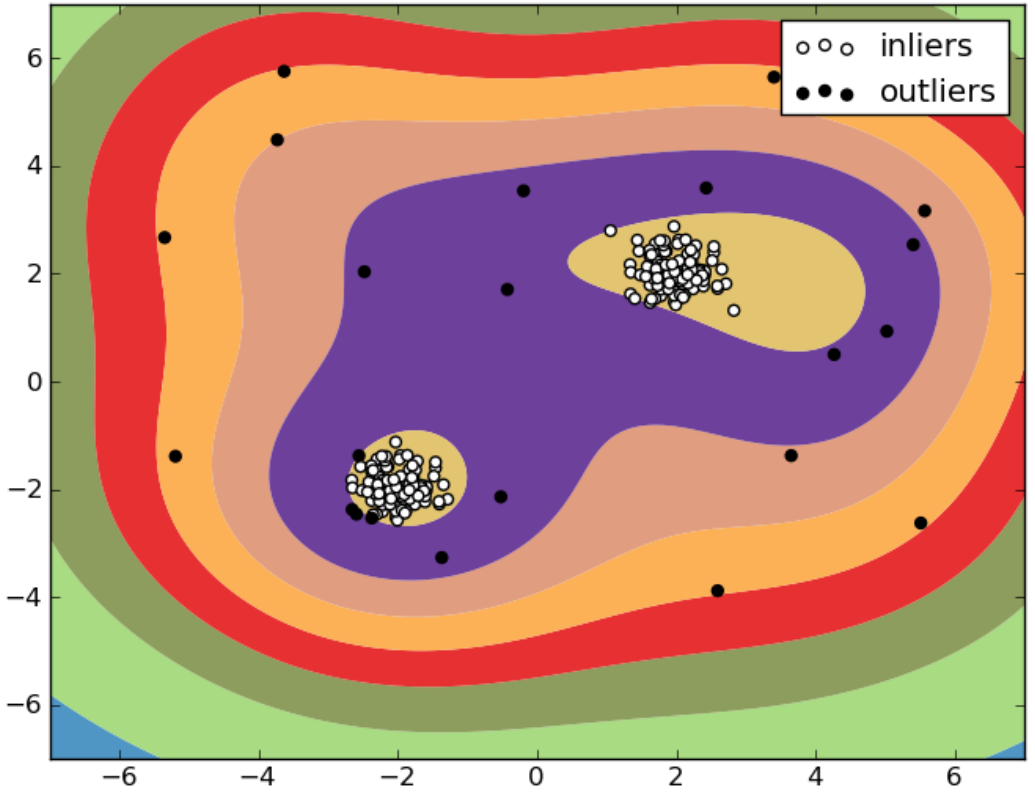


One-class SVM with non-linear mapping

$$\min_{r,c} r^2$$

$$\|\Phi(\boldsymbol{x}_i) - c\|^2 \leq r^2, \forall i = 1, \ldots, N$$

where $\Phi(\cdot)$: $\mathbb{R}^p \rightarrow \mathbb{R}^q$ is a non-linear mapping



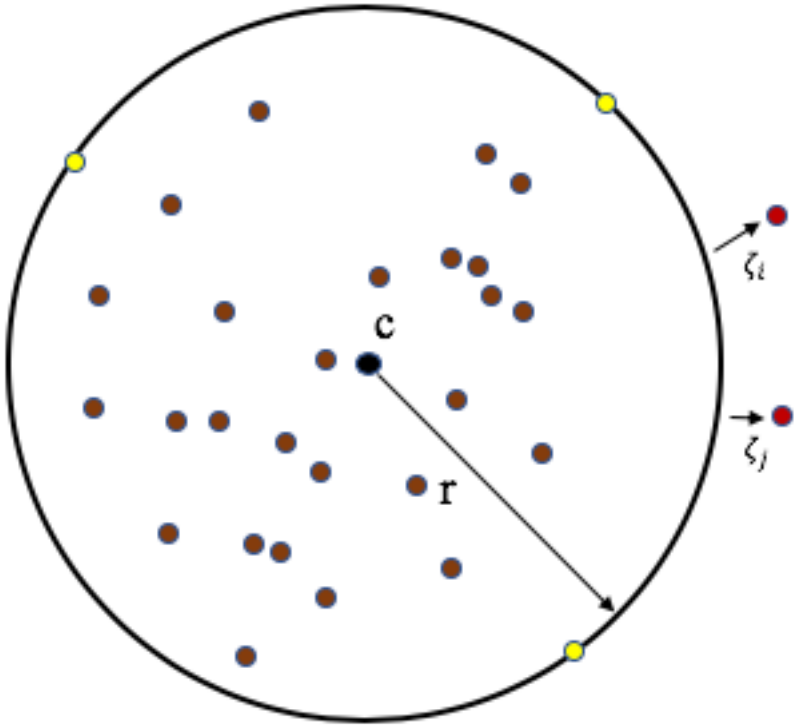
Problem statement with slack variables

- SVM based OCC with slack variables relies on identifying the smallest hypersphere (with radius $r \geq 0$, and center $c \in \mathbb{R}^q$) where some samples are allowed to fall out of the hypersphere.

$$\min_{r,c,\zeta_i} r^2 + \frac{1}{\nu N}\sum_{i=1}^{N}\zeta_i$$

$$\|\Phi(\boldsymbol{x}_i) - c\|^2 \leq r^2 + \zeta_i, \forall i = 1, \ldots, N$$
$$\zeta_i \geq 0$$

for a dataset $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$ and the set of slack variables $\{\zeta_i\}_{i=1}^N$



Isolation Forest

Isolation Forest

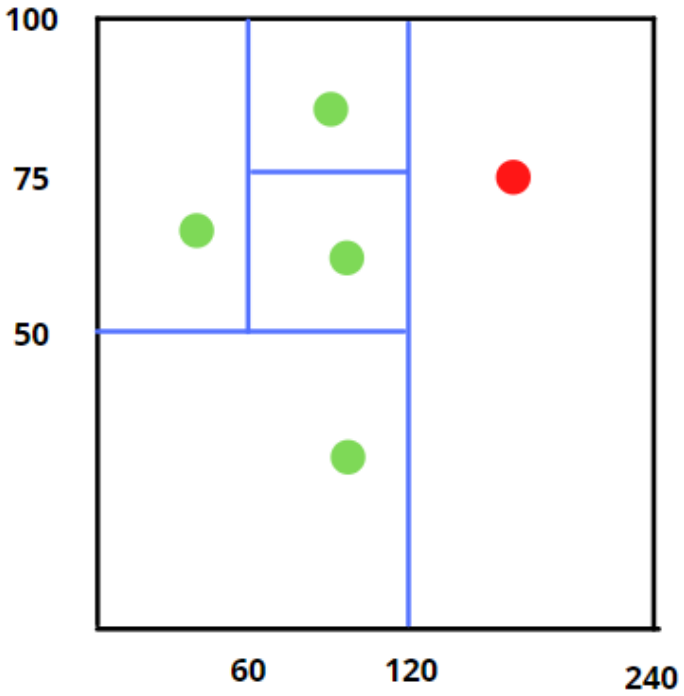
- Isolation forest works on the principle of the decision tree algorithm.
- It isolates the outliers by randomly selecting a feature from the given set of features and then randomly selecting a split value between the maximum and minimum values of the selected feature.
- This random partitioning of features will produce smaller paths in trees for the anomalous data values and distinguish them from the normal set of the data.

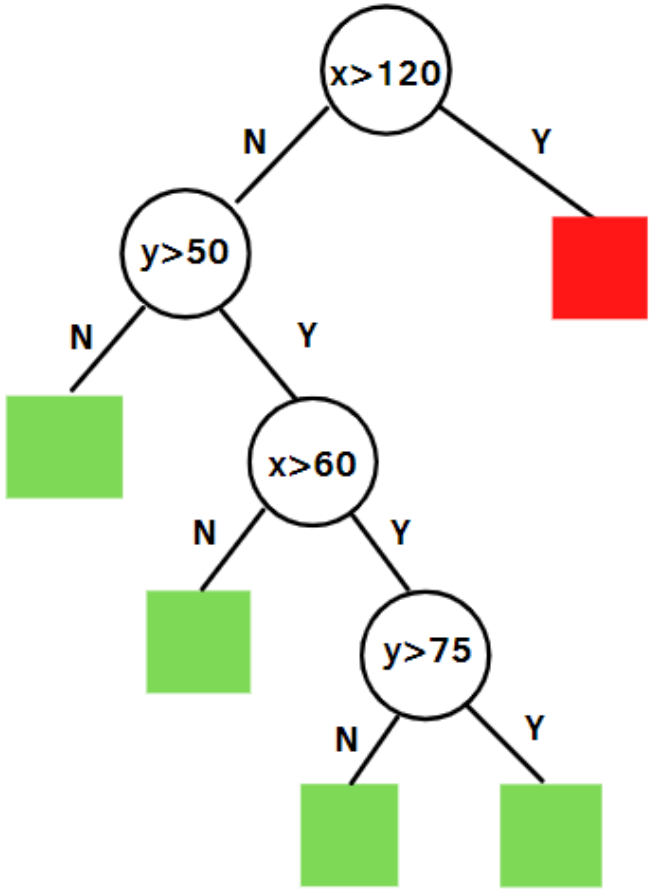
Algorithm

- Let $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^{N}$ be a set of p-dimensional points
- An Isolation Tree (iTree) is defined as a data structure with the following properties:
  - For each node E in the tree, E is either an external-node with no child, or an internal-node with one « test » and exactly two daughter nodes (E_l and E_r)
  - A test at node E consists of an attribute A and a split value v such that the test $A < v$ determines the traversal of a data point to either E_l or E_r.

- In order to build an iTree, the algorithm recursively divides $\mathcal{D}$ by randomly selecting an attribute A and a split value v, until either
  - The node has only one instance, or
  - All data at the node have the same values.

- Assuming all instances are distinct, each instance is isolated to an external node when an iTree is fully grown, in which case the number of external nodes is N and the number of internal nodes is $N - 1$





https://betterprogramming.pub/anomaly-detection-with-isolation-forest-e41f1f55cc6

Anomaly detection

- The task of anomaly detection is to provide a ranking that reflects the degree of anomaly.
- Thus, one way to detect anomalies is to sort data points according to their path lengths or anomaly scores.
  - **Anomalies** are points that are ranked at the top of the list.

- **Path Length** $h(\boldsymbol{x})$ of a point $\boldsymbol{x}$ is measured by the number of edges $\boldsymbol{x}$ traverses an iTree from the root node until the traversal is terminated at an external node.

- The difficulty in deriving an anomaly score from $h(\boldsymbol{x})$ is that, while the maximum possible height of iTree grows in the order of N, the average height grows in the order of $\log N$. A relevant normalization of $h(\boldsymbol{x})$ is necessary to get a useful anomaly score.

Liu, Fei Tony; Ting, Kai Ming; Zhou, Zhi-Hua. "Isolation Forest". *2008 Eighth IEEE International Conference on Data Mining*: 413–422

Anomaly score

- iTrees have an equivalent structure to Binary Search Tree (BST)
- The estimation of average $h(\boldsymbol{x})$ for external node terminations is the same as the unsuccessful search in BST.
- Given a data set of N instances, the average path length of unsuccessful search in BST as:

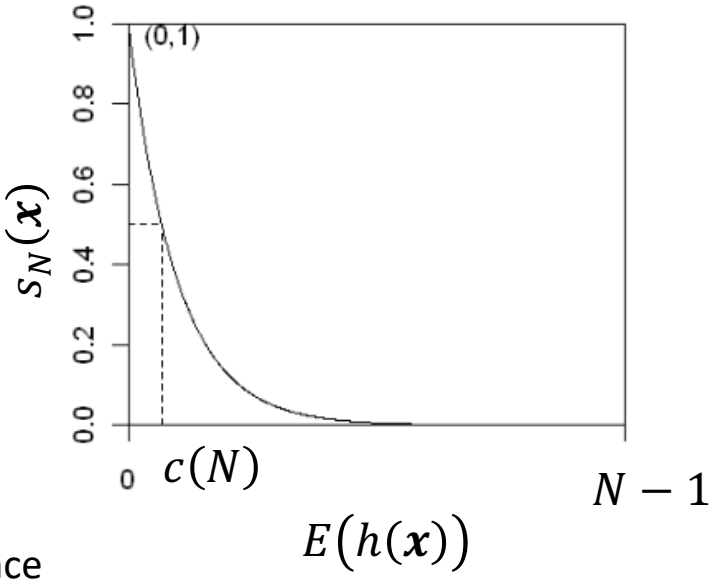
$$c(N) = 2H(N-1) - \frac{2(N-1)}{N}$$

where $H(i)$ is the harmonic number

- The anomaly score $s_N(\boldsymbol{x})$ of an instance $\boldsymbol{x}$ is defined as:

$$s_N(\boldsymbol{x}) = 2^{-\frac{E(h(\boldsymbol{x}))}{c(N)}}$$

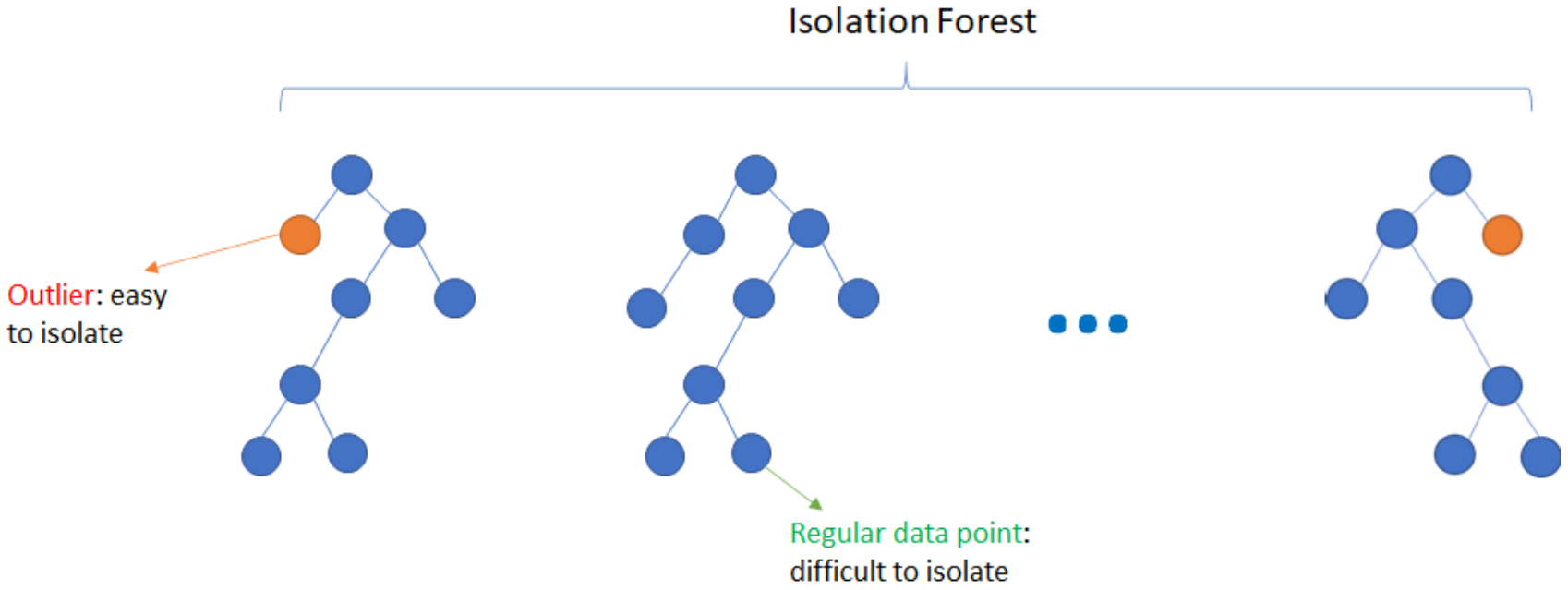
  - $E(h(\boldsymbol{x}))$ is the average of $h(\boldsymbol{x})$ from a collection of isolation trees.
  - $0 \leq s_N(\boldsymbol{x}) \leq 1$:
    - if $s_N(\boldsymbol{x})$ is very close to 1, then $\boldsymbol{x}$ is definitely an anomaly,
    - if $s_N(\boldsymbol{x})$ is much smaller than 0.5, then it is safe to be regarded as a normal instance



https://en.wikipedia.org/wiki/Binary_search_tree

Isolation forest

- A single isolation tree has a lot of expected variability in the isolation depths that it will give to each observation.
- Thus an ensemble of many such trees - an “isolation forest” - may be used instead for better results, with the final score obtained by averaging the results (the isolation depths) from many such trees.

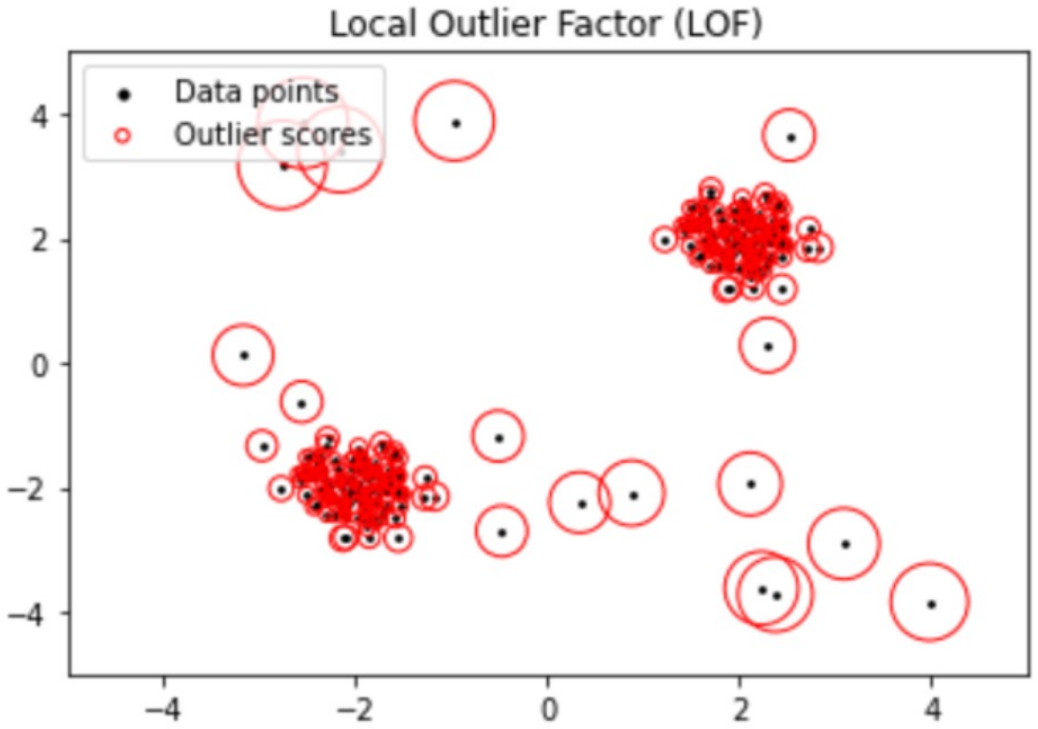


https://engineering.teknasyon.com/what-are-isolation-forests-151d8e98ef5f

Local Outlier Factor

Local Outlier Factor

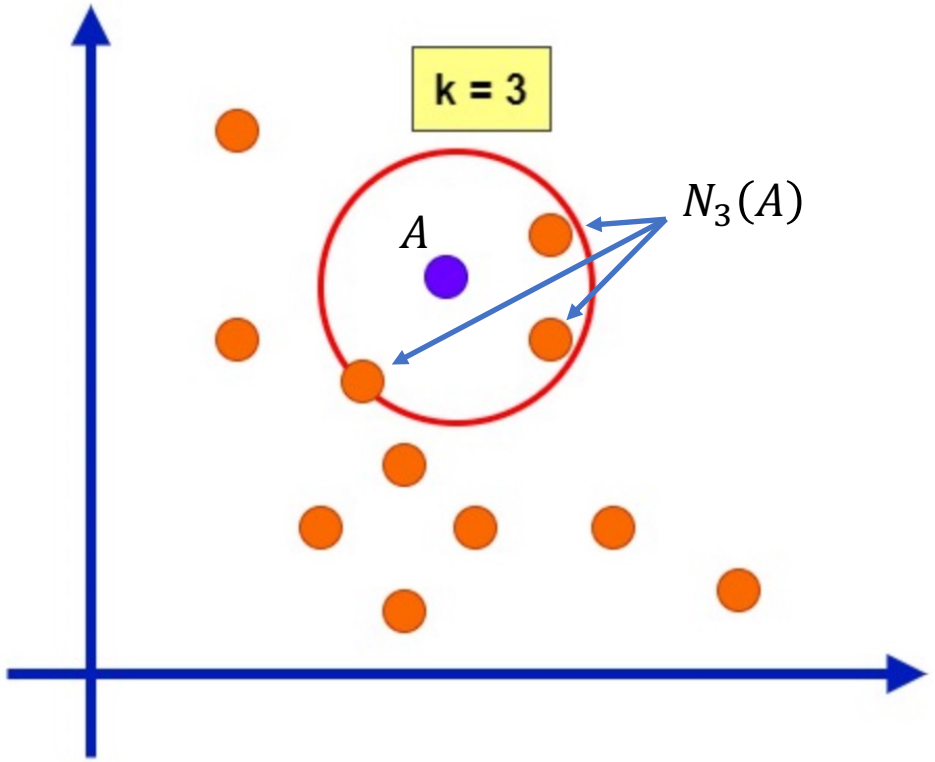
- The Local Outlier Factor (LOF) is a density-based outlier detection method derived from DBSCAN
- Intuition behind the approach : the density around an outlier object will be significantly different from the density around its neighbors.
- It considers as outliers the samples that have a substantially lower density than their neighbours.



https://www.geeksforgeeks.org/local-outlier-factor/

K-Distance

- Let $d(A, B)$ be a distance between two objects A and B
- Let *k–distance*(A) be the distance of the object A to the k-th nearest neighbor.
- Note that the set of the k nearest neighbors includes all objects at this distance, which can in the case of a "tie" be more than k objects.
- We denote the set of k nearest neighbors as $N_k(A)$.

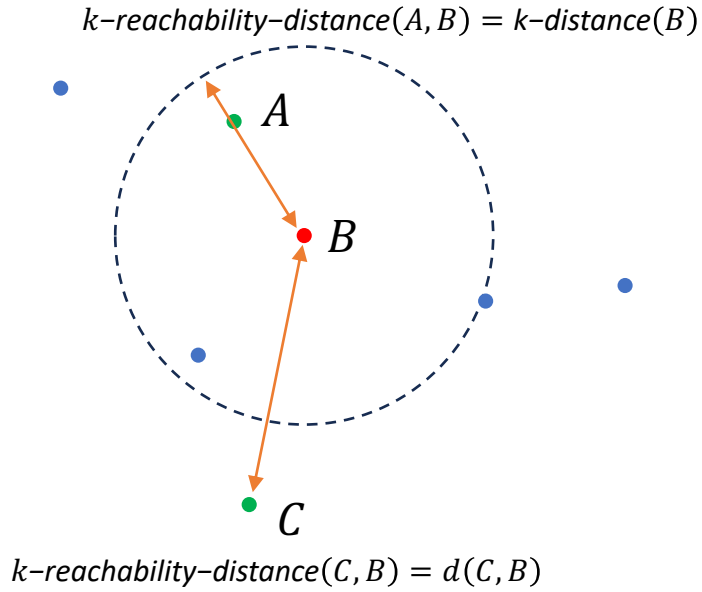


K-Reachability Distance

- This distance is used to define what is called reachability distance:

  $k\text{–}reachability\text{–}distance(A, B) = \max\{k\text{–}distance(B), d(A, B)\}$

  - The reachability distance of an object A from B is the true distance of the two objects, but at least the *k–distance* of B.
  - Objects that belong to the k nearest neighbors of B (the "core" of B) are considered to be equally distant.
  - Increasing the value for k increases the smoothing effect.
  - Note that this is not a distance in the mathematical definition, since it is not symmetric.



Local Reachability Density

- The local reachability density $k\text{–}lrd(A)$ of an object A is defined by

$$k\text{–}lrd(A) = \frac{1}{\frac{\sum_{B \in N_k(A)} k\text{–}reachability\text{–}distance(A,B)}{|N_k(A)|}}$$

- This is the inverse of the average reachability distance of the object A from its neighbors.
- This is the distance at which A can be "reached" from its neighbors.

Local Outlier Factor Score

- The local reachability densities are then compared with those of the neighbors using

$$k\text{–}LOF(A) = \frac{\sum_{B \in N_k(A)} \frac{k\text{–}lrd(B)}{k\text{–}lrd(A)}}{|N_k(A)|} = \frac{\sum_{B \in N_k(A)} k\text{–}lrd(B)}{|N_k(A)|\ k\text{–}lrd(A)}$$

- This is the average local reachability density of the neighbors divided by the object's own local reachability density.
- Interpretation
  - $k\text{–}LOF(A) \approx 1$ means Similar density as neighbors,
  - $k\text{–}LOF(A) < 1$ means Higher density than neighbors (Inlier)
  - $k\text{–}LOF(A) > 1$ means Lower density than neighbors (Outlier)

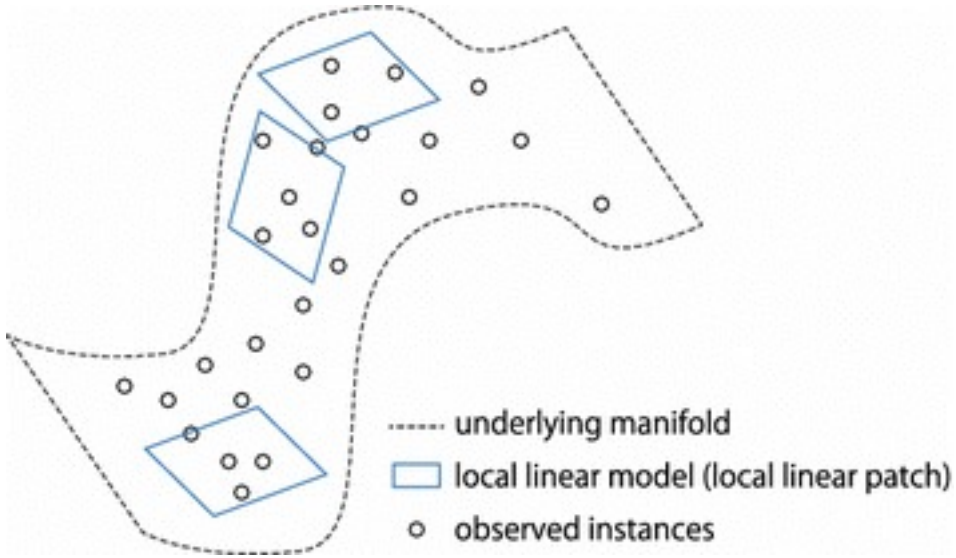
Manifold Learning

Main methods

- Multi-dimensional Scaling (MDS)
- Isomap
- Locally Linear Embedding (LLE)
- Hessian Eigenmapping (Hessian-based LLE or HLLE)
- Spectral Embedding
- Local tangent space alignment (LTSA)
- t-distributed Stchastic Neighbor Embedding (t-SNR)

Locally Linear Embedding (LLE)

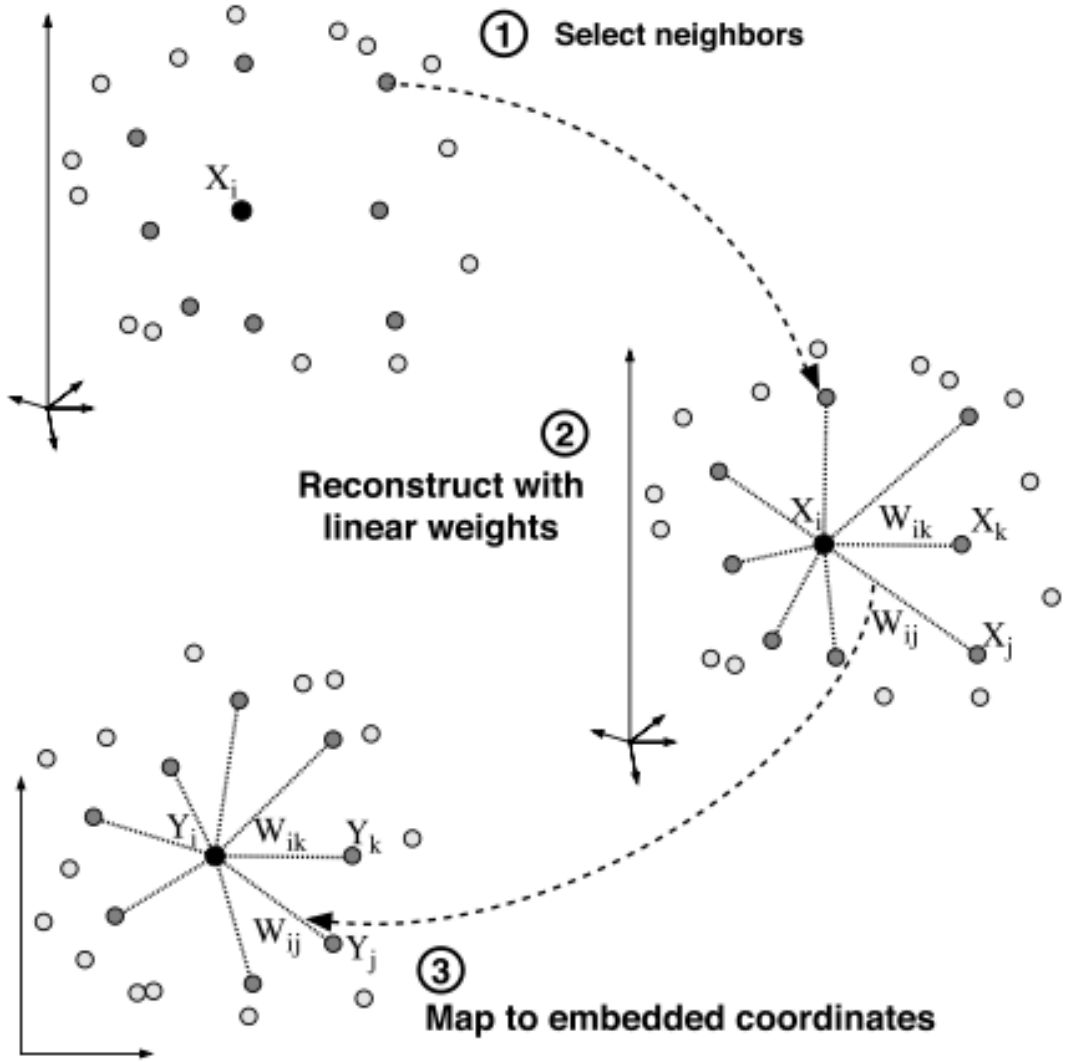
- Manifold Characteristics/Key Assumption
  - We expect each data point and its neighbors to lie on or close to a locally linear patch
  - But, how to combine all local patches together?



LLE Intuition

- Assume that manifold is approximately « linear » when viewed locally (in a small neighborhood)
- A good projection should preserve this local geometric property as much as possible

LLE Algorithm Steps



LLE Algorithm

- Step1: Select neighbors for each data instance $\boldsymbol{x}_i \in \mathbb{R}^p$
- Step 2: Each data instance is written as a convex combination of its neighbors. The weights of the « convex » combination « reconstruct » each point from its neighbors. The weights chosen aim to minimize the reconstruction error

$$J(W) = \sum_{i=1}^{N} \left\| \boldsymbol{x}_i - \sum_{j=1}^{N} W_{ij} \boldsymbol{x}_j \right\|_2^2$$

where

- $W_{ij} = 0$ if $\boldsymbol{x}_j$ does not belong to set of neighbors of $\boldsymbol{x}_i$ (for example, $W_{ii} = 0$)
- The rows of the weight matrix sum to one i.e. $\sum_{j=1}^{N} W_{ij} = 1$
- The weights can be negative.

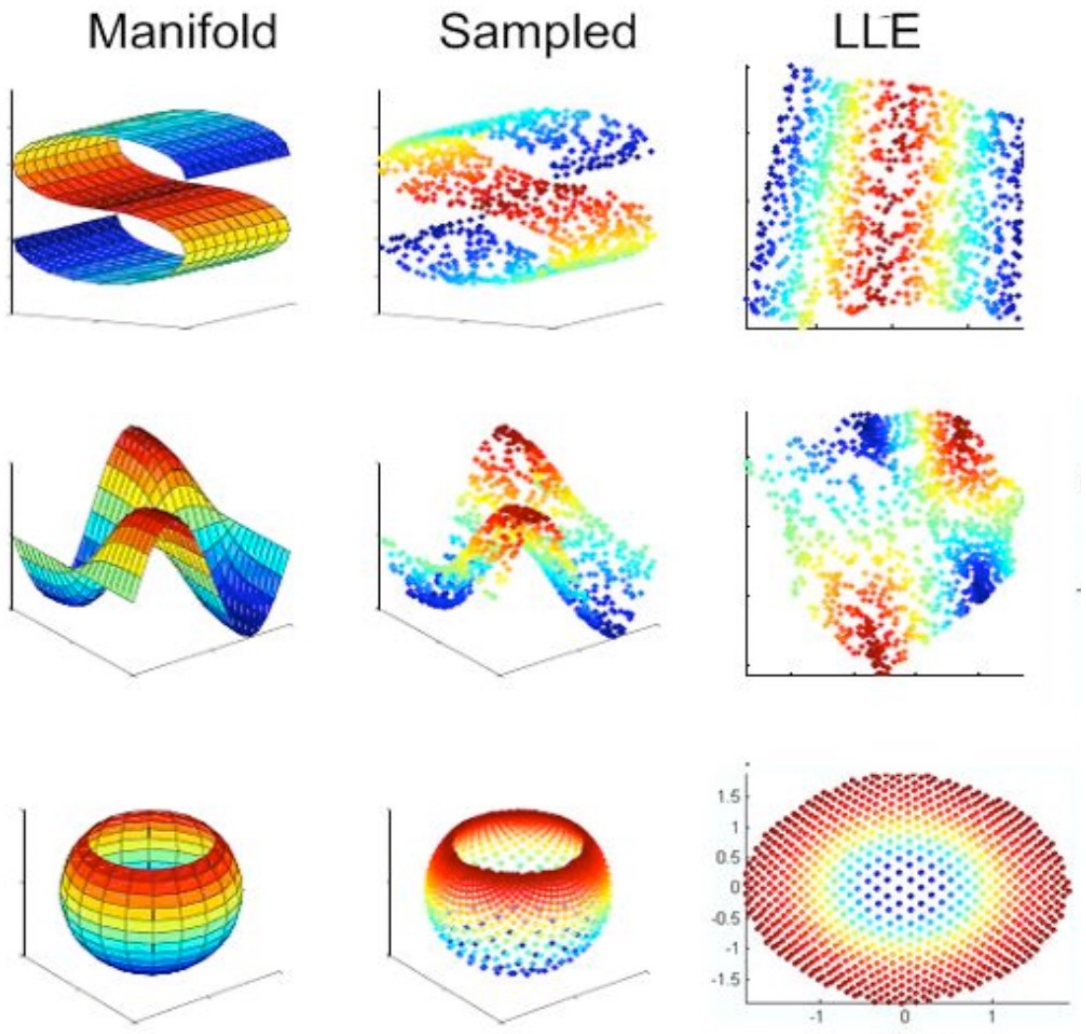
LLE Algorithm

- Step3: Map $\mathbb{R}^p$ to a low-dimensional embedding $\mathbb{R}^k$: $\boldsymbol{x}_i \in \mathbb{R}^p \rightarrow \boldsymbol{y}_i \in \mathbb{R}^k$
- The cost function can be minimized by solving a sparse eigenvalue problem

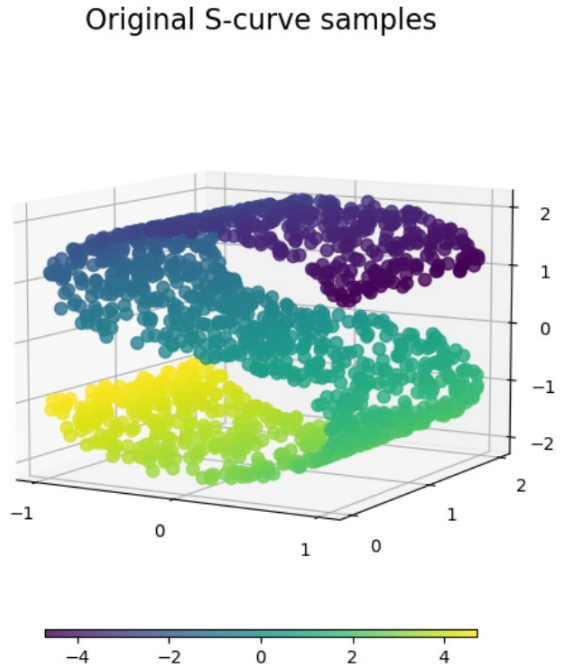
$$J(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N) = \sum_{i=1}^{N} \left\| \boldsymbol{y}_i - \sum_{j=1}^{N} W_{ij} \boldsymbol{y}_j \right\|_2^2$$

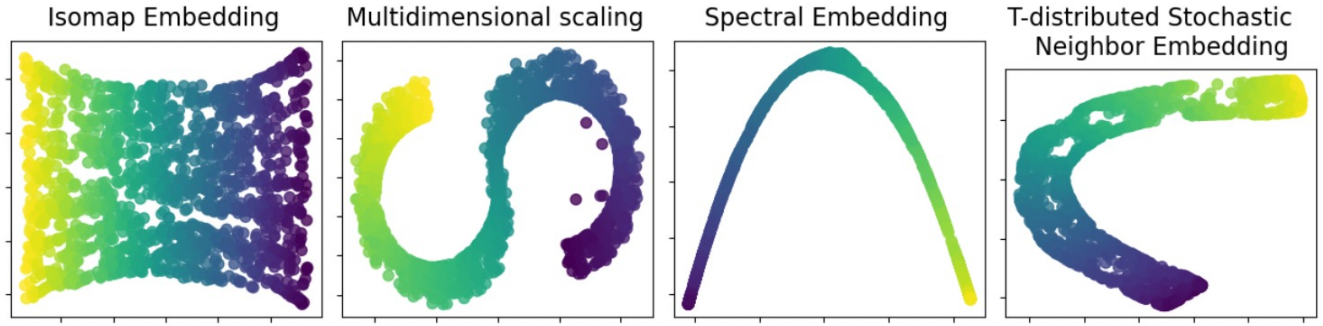
- Note:
  - The weights are constant here. We compute only the $\boldsymbol{y}_i$'s!
  - The $\boldsymbol{y}_i$'s represent global internal coordinates on the manifold
  - Some additional constraints are desirable to give better results
  - $J(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N)$ is a quadratic form => convex optimization!

LLE Example ($p = 2\,, k = 2$)



Other manifold learning algorithms





Conclusion

Conclusion

- Unsupervised learning is very challenging but quite appealing in data science!
- It is generally difficult/expensive to get labelled datasets
- Unsupervised methods can deal with classification or estimation
- Non-linearity should be processed very carefully!