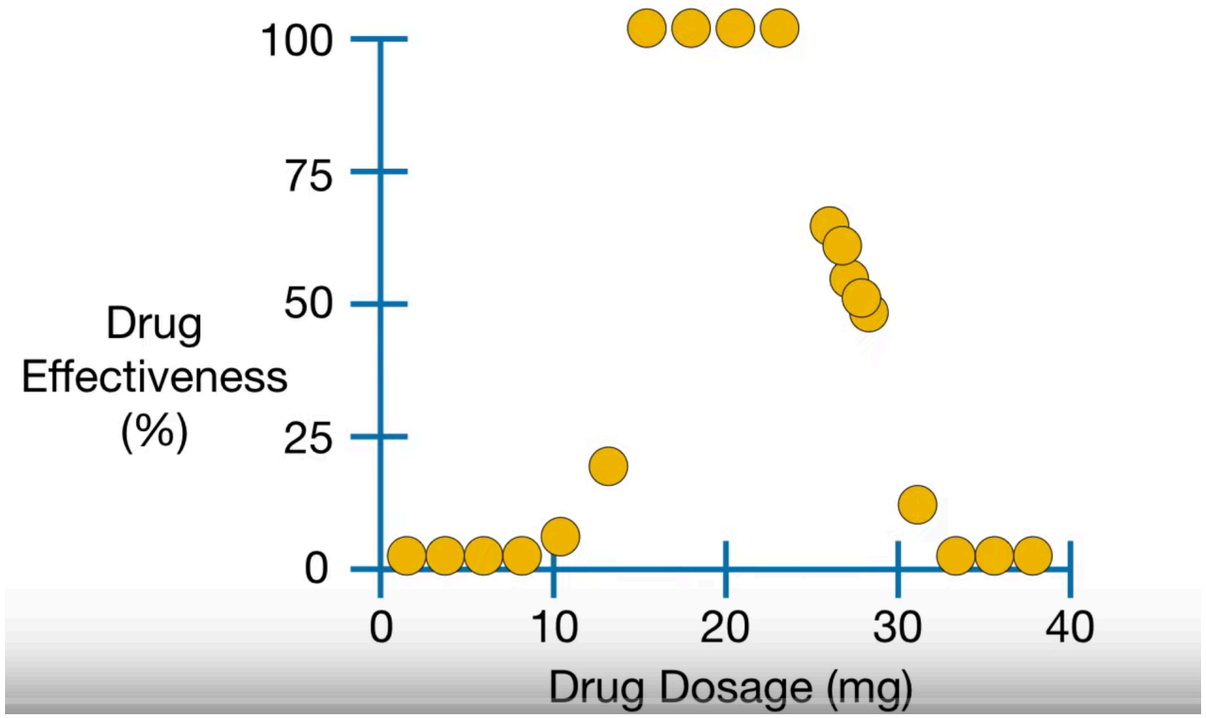
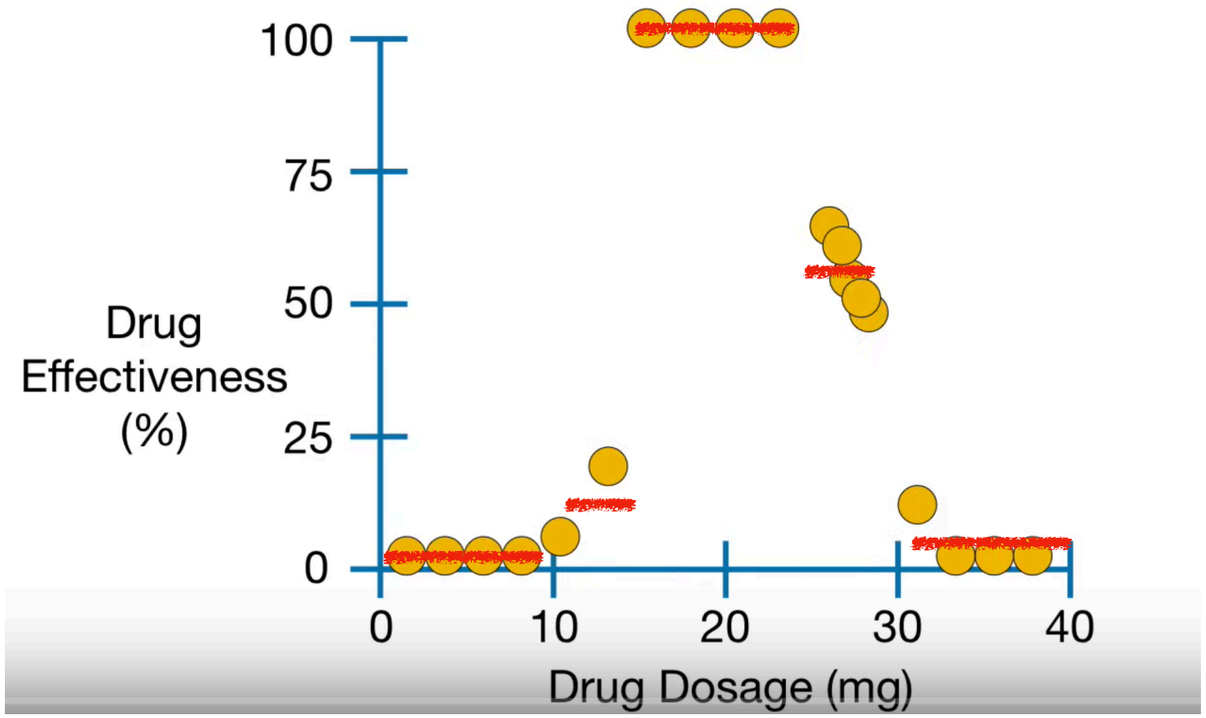
Introduction to Machine Learning

Part 3.

Hamid Mirisaee, Vera Shalaeva

Regression Tree

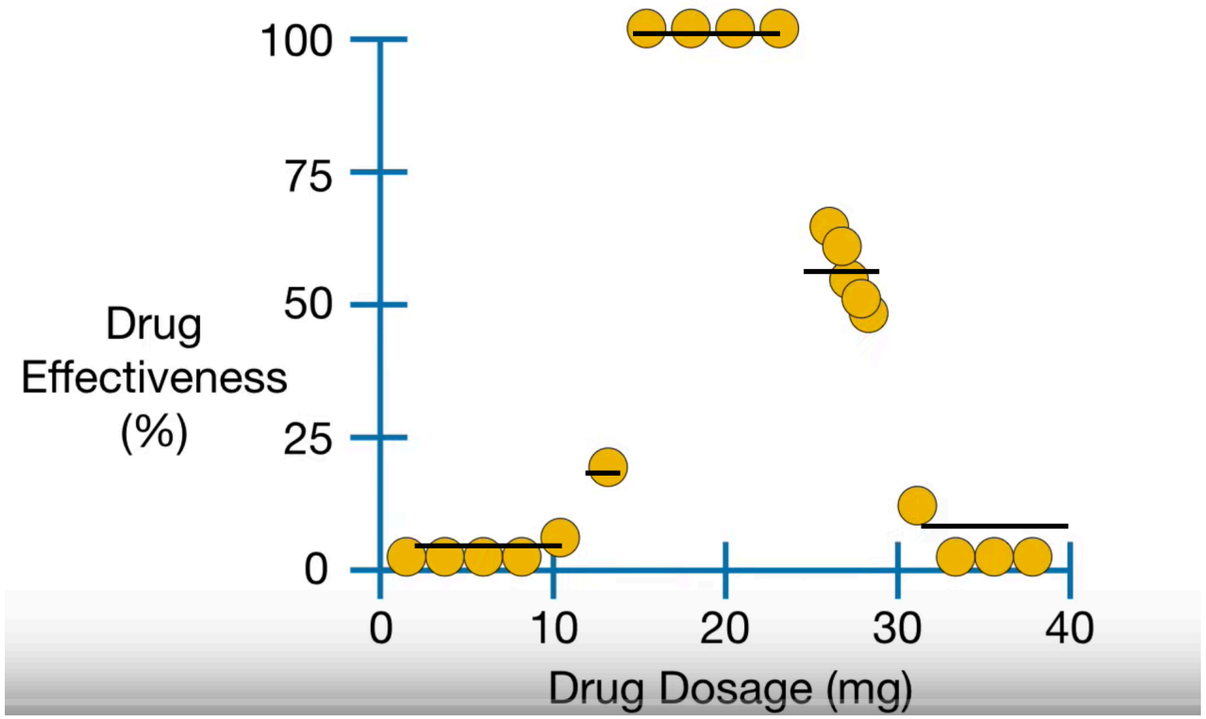


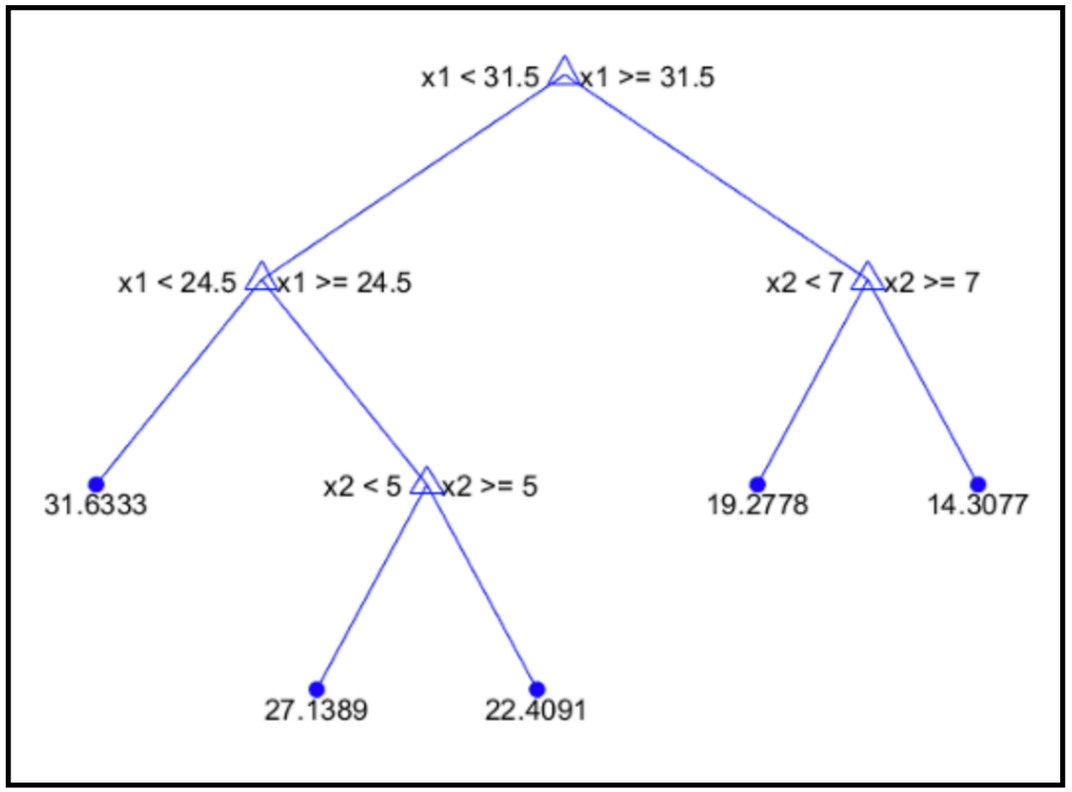


Regression Tree

- Not always possible to model the data with LR.
- Regression Trees model the data using the discriminative property of attributes (you may wanna watch this).
- Based on a top-down, recursive splitting approach.
- If we split too much —> overfitting.
    - One prediction per example.
- Stop splitting a node when reaching a minimum number of examples at the node (can be determined with k-fold cross validation).
- Regression trees are simple to build and easy to understand.
- Plus, it can give you a good idea of important features.

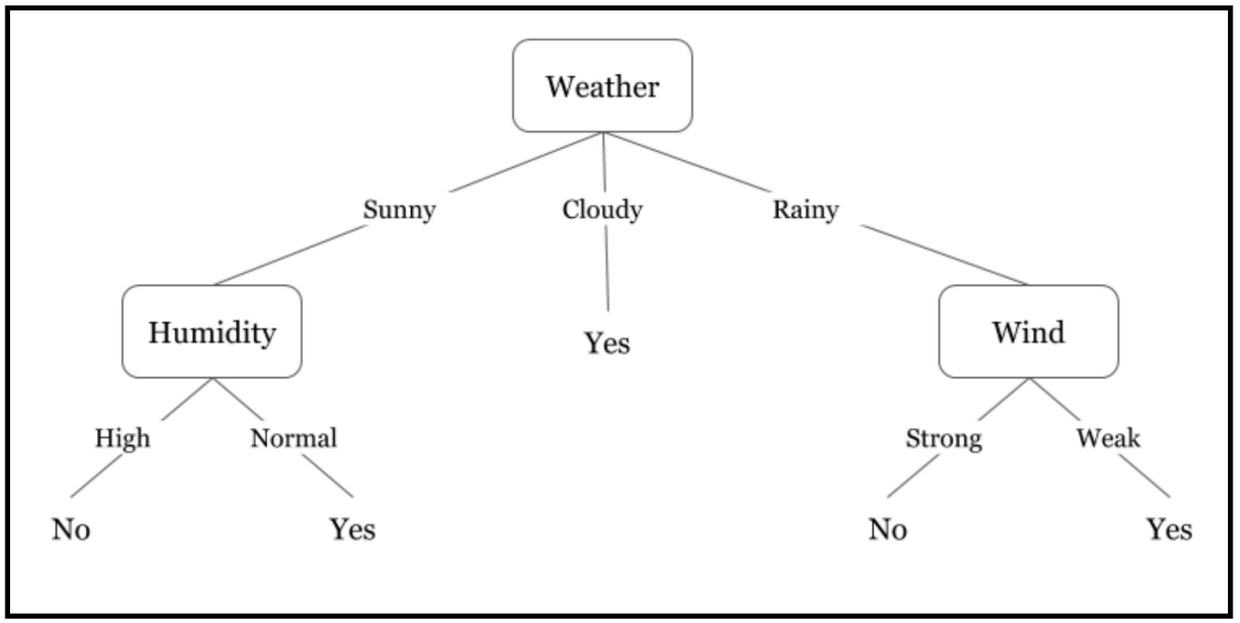
Decision Tree





- Remember Regression Tree?
    - Group the data in the form of a tree, predict value for each group.
    - Each leaf represents a certain value for the prediction.
    - If each leaf == one example —> exact prediction —> overfitting.
        - Solution 1: keep a min #instances for nodes, after that, do not split.
        - Solution 2: keep a max depth for the tree.
- Decision Tree is almost the same thing (and usually top-down):
    - Each leaf predicts one particular class instead of one particular value.
    - The principle is the same: divide based on the attribute/value which makes the most distinction.

Decision Tree



- Note that each leaf represents only one class.
- In a binary classification —> some of the leafs represent the positive class and others the negative class.
- Pros:
    - Easy to build/understand.
    - Can handle numerical AND categorical data.
    - Built-in feature selection method.
- Cons:
    - Not robust, small changes in data —> a completely different tree.
    - Biased towards categorical data with multiple values.
    - It is NP-complete.

Decision Tree

- How do we build a Decision Tree?
    - Different metics: Gini impurity, Information Gain, etc.
- We see here one of them: Information Gain (IG).
- Which attribute should we choose for the first split?
    - The one that if used, it results in decreasing the entropy of the dataset.
    - Entropy —> the amount of uncertainty.
    - We choose the attribute which makes the dataset the purest.
    - To do that, we need to *asses* each attribute.
    - We need to calculate its IG.
    - For a give attribute a —> $\overbrace{IG(T,a)}^{\text{Information Gain}} = \overbrace{\mathrm{H}(T)}^{\text{Entropy (parent)}} - \overbrace{\mathrm{H}(T|a)}^{\text{Sum of Entropy (Children)}}$

Decision Tree: entropy

Entropy

$$\mathrm{H}(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log_b \mathrm{P}(x_i)$$

Conditional Entropy

$$\mathrm{H}(X|Y) = -\sum_{i,j} p(x_i, y_j) \log \frac{p(x_i, y_j)}{p(y_j)}$$

Decision Tree: IG

Information Gain for an attribute

$$\overbrace{IG(T,a)}^{\text{Information Gain}} = \overbrace{\mathrm{H}(T)}^{\text{Entropy (parent)}} - \overbrace{\mathrm{H}(T|a)}^{\text{Sum of Entropy (Children)}}$$

$$= -\sum_{i=1}^{J} p_i \log_2 p_i - \sum_{i=1}^{J} -\Pr(i|a) \log_2 \Pr(i|a)$$

Entropy of the data MINUS the conditional entropy of the attribute

Decision Tree: example from this video

outlook	temperature	humidity	wind	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cold	normal	false	yes
rainy	cold	normal	true	no
overcast	cold	normal	true	yes
sunny	mild	high	false	no
sunny	cold	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

$$\mathrm{H}(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log_b \mathrm{P}(x_i)$$

PLAYING GOLF
→ **9** times **YES**
→ **5** times **NO**

We just have to use the Shannon-entropy formula to calculate the **H(x)** values

H(PlayingGolf) = H(9,5) =

= -(0.64 log$_2$0.64) – (0.36 log$_2$0.36) = 0.94

$\frac{9}{14}$ $\frac{9}{14}$ $\frac{5}{14}$ $\frac{5}{14}$

Entropy (impurity) of the entire data

Decision Tree: example from this video

outlook	temperature	humidity	wind	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cold	normal	false	yes
rainy	cold	normal	true	no
overcast	cold	normal	true	yes
sunny	mild	high	false	no
sunny	cold	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

H(PlayingGolf) = H(9,5) =

$$= -(0.64 \log_2 0.64) - (0.36 \log_2 0.36) = 0.94$$

$\frac{9}{14}$ $\frac{9}{14}$ $\frac{5}{14}$ $\frac{5}{14}$

$$E(T,X) = \sum_x P(x)\, E(x)$$

We have to calculate the entropy with respect to a given predictor/feature in order to be able to calculate information gain

		PLAY GOLF YES	PLAY GOLF NO	
OUTLOOK	sunny	2	3	
	overcast	4	0	→ No impurity
	rainy	3	2	

E(PlayGolf,Outlook) = P(sunny)E(2,3) + P(overcast)E(4,0) + P(rainy)E(3,2)

$$\frac{5}{14}\,0.971 + \frac{4}{14}\,0 + \frac{5}{14}\,0.971 = 0.6936$$

Probability of sunny outlook in the dataset

Entropy of the data when the outlook is sunny

$$-(\frac{2}{5}log\frac{2}{5}) - (\frac{3}{5}log\frac{3}{5})$$

Decision Tree: example from this video

outlook	temperature	humidity	wind	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cold	normal	false	yes
rainy	cold	normal	true	no
overcast	cold	normal	true	yes
sunny	mild	high	false	no
sunny	cold	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

H(PlayingGolf) = H(9,5) =

$$= -(0.64 \log_2 0.64) - (0.36 \log_2 0.36) = 0.94$$

$\frac{9}{14}$ $\frac{9}{14}$ $\frac{5}{14}$ $\frac{5}{14}$

$$E(T,X) = \sum_x P(x)\, E(x)$$

We have to calculate the entropy with respect to a given predictor/feature in order to be able to calculate information gain

		PLAY GOLF YES	PLAY GOLF NO
OUTLOOK	sunny	2	3
	overcast	4	0
	rainy	3	2

E(PlayGolf,Outlook) = P(sunny)E(2,3) + P(overcast)E(4,0) + P(rainy)E(3,2)

$$\frac{5}{14}\,0.971 + \frac{4}{14}\,0 + \frac{5}{14}\,0.971 = 0.6936$$

Information Gain = H(PlayGolf) - E(PlayGolf,Outlook) =
= 0.94 – 0.693 = 0.247

Decision Tree: example from this video

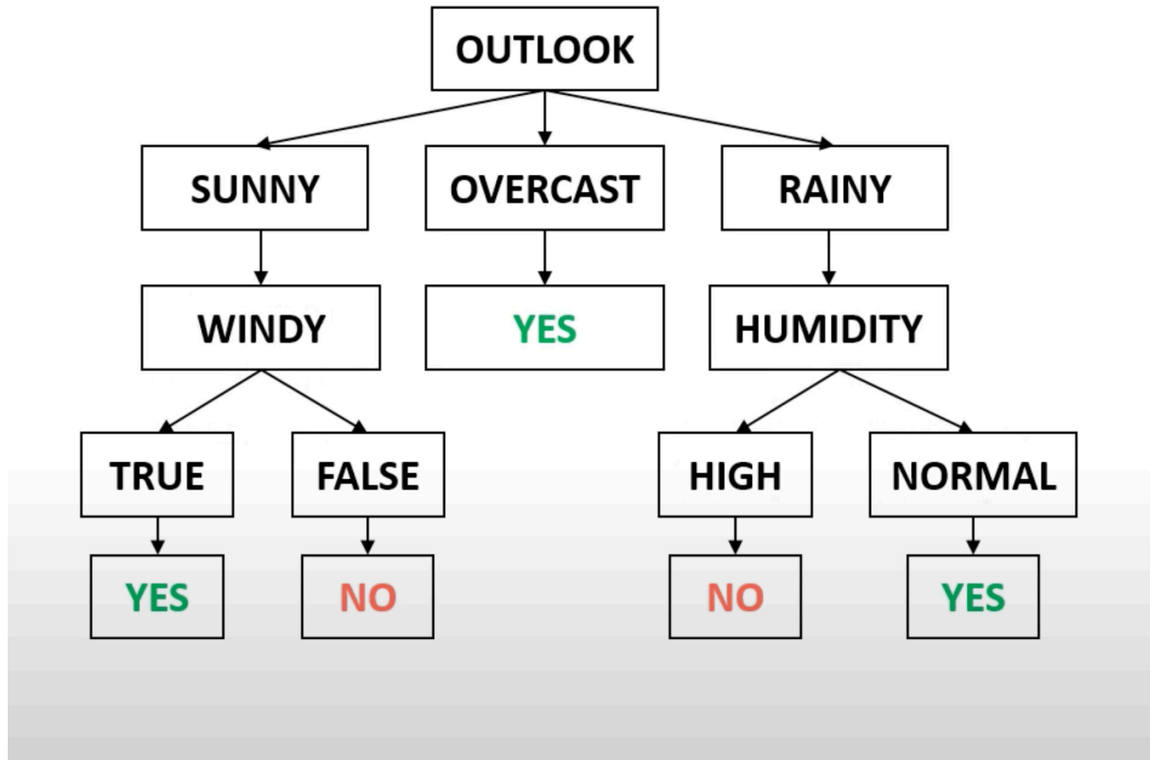
outlook	temperature	humidity	wind	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cold	normal	false	yes
rainy	cold	normal	true	no
overcast	cold	normal	true	yes
sunny	mild	high	false	no
sunny	cold	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

Information Gain (outlook) = 0.247

Information Gain (temperature) = 0.029

Information Gain (humidity) = 0.152

Information Gain (wind) = 0.048



Ensemble Learning

- General definition: use multiple models to get better performance (wiki).
- Bagging (**B**ootstrap **agg**regat**ing**) —> mostly for Decision Trees (Random Forest).
    - Given the training set, create *m* subsets by randomly pick the instances.
        - Selection WITH replacement [why?] —> in a subset we may have duplicates.
    - New instance —> use all models via averaging (regression) or voting (classification).
- Boosting: use a set of weak learners to build a strong learner.
    - How does it work?
        1. Learn a classifier.
        2. Identify the misclassified examples.
        3. Reweigh the data using them.
        4. Learn another classifier.
        5. Compute the importance in taking the decision.
        6. Decide!
    - AdaBoost is probably the most known boosting method. Watch this video.

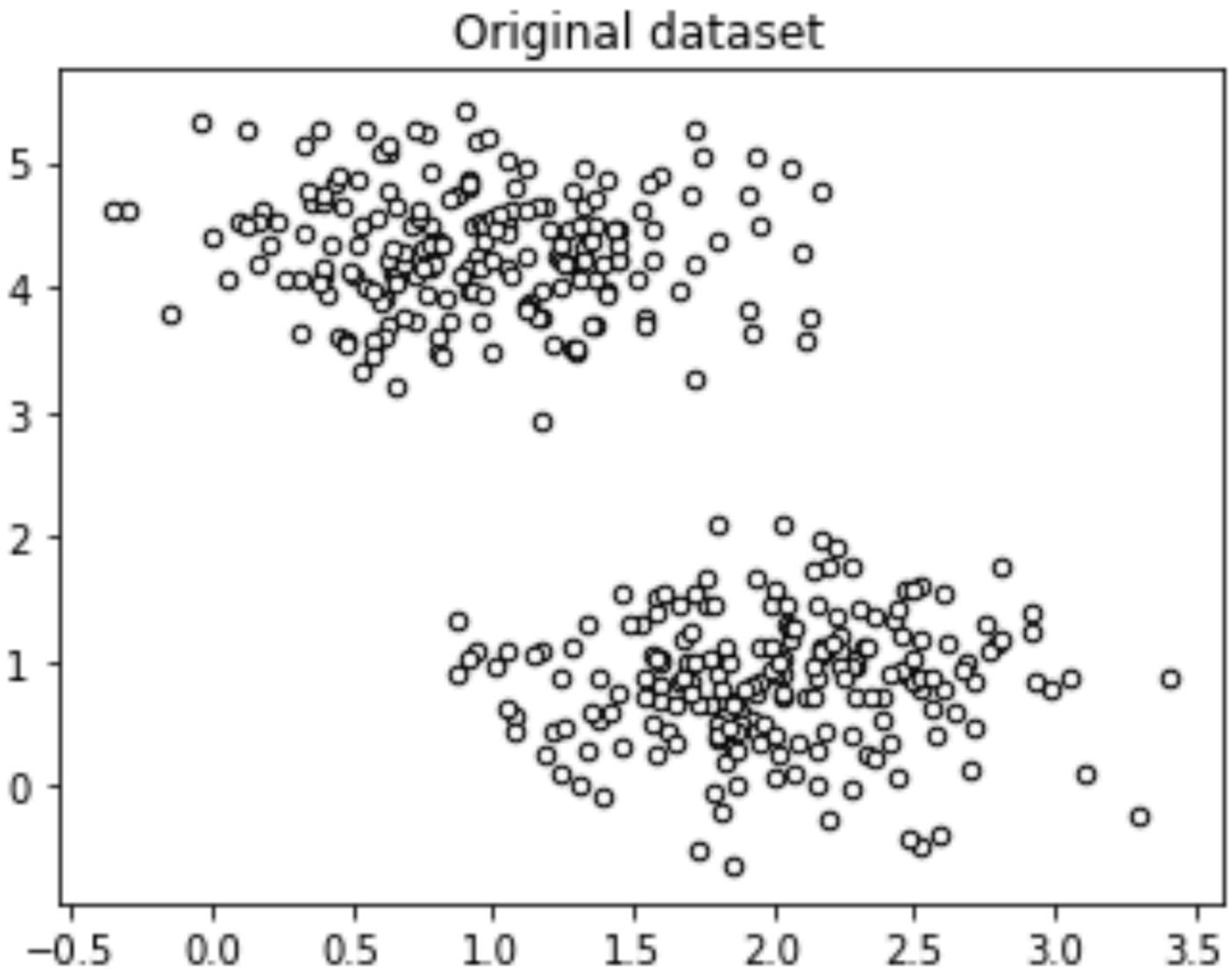
So far…

- Supervised learning —> when we have data with known labels or know values to be predicted.
    - Regression —> predicting value.
        - Linear Regression, Regression Tree.
    - Classification —> classifying data, predicting class.
        - Logistic Regression, Decisions Tree.
        - Ensemble learning:
            - Bagging (ex: Random Forest) , boosting (ex: AdaBoost).

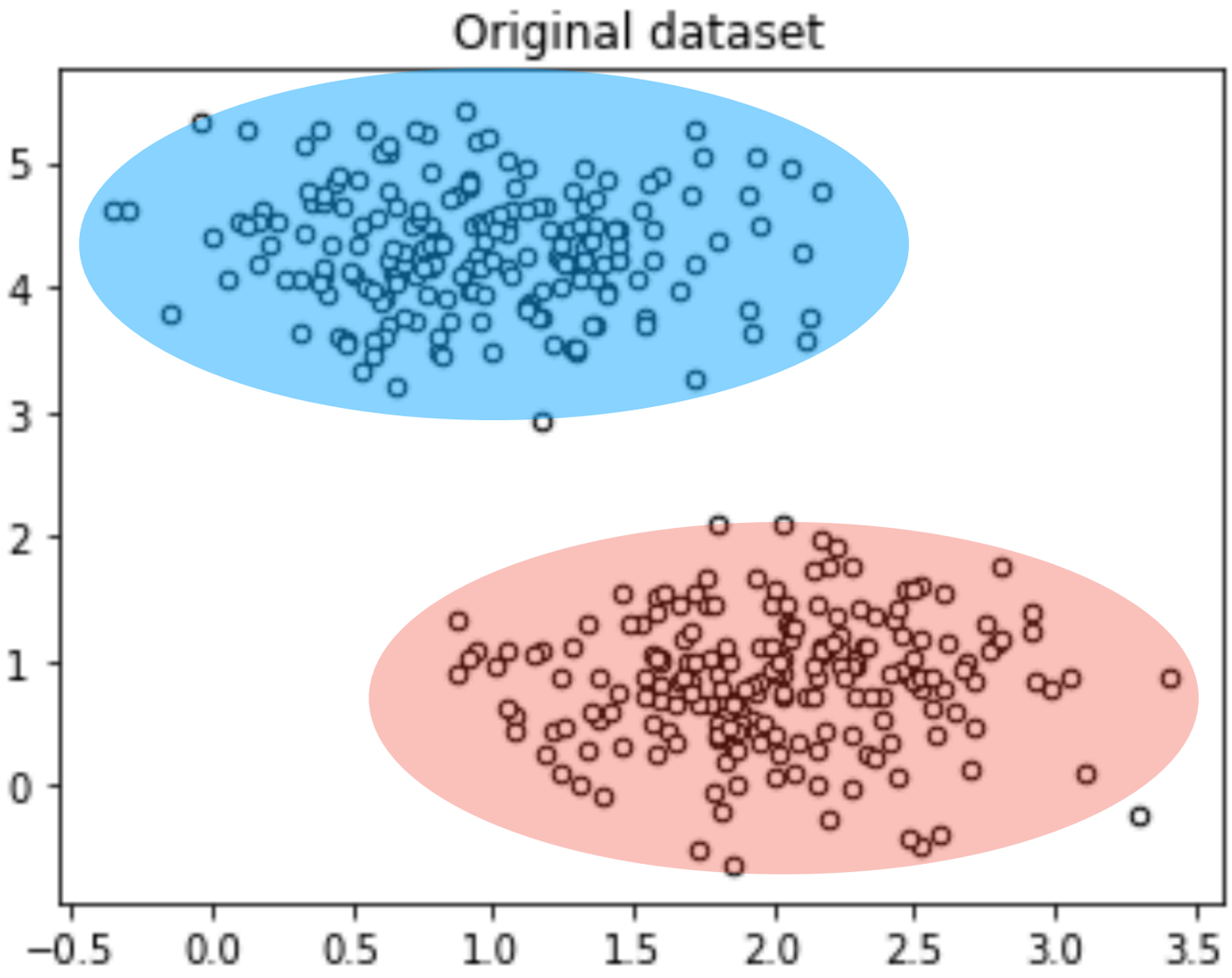
So far…

- Supervised learning —> when we have data with known labels or know values to be predicted.
    - Regression —> predicting value.
        - Linear Regression, Regression Tree.
    - Classification —> classifying data, predicting class.
        - Logistic Regression, Decisions Tree.
        - Ensemble learning:
            - Bagging (ex: Random Forest) , boosting (ex: AdaBoost).
- Unsupervised learning —> most of the times, we do not have the labels!
    - Why? Because observation is easy (is it?!) but annotation is very costly.
    - Clustering, (some) neural networks, anomaly detection, etc.
    - Clustering —> grouping the data point such that similar instances are together.
        - Similar? How to define the similarity?
        - Kmeans, Kmediods, hierarchical clustering, DBSCAN, etc.

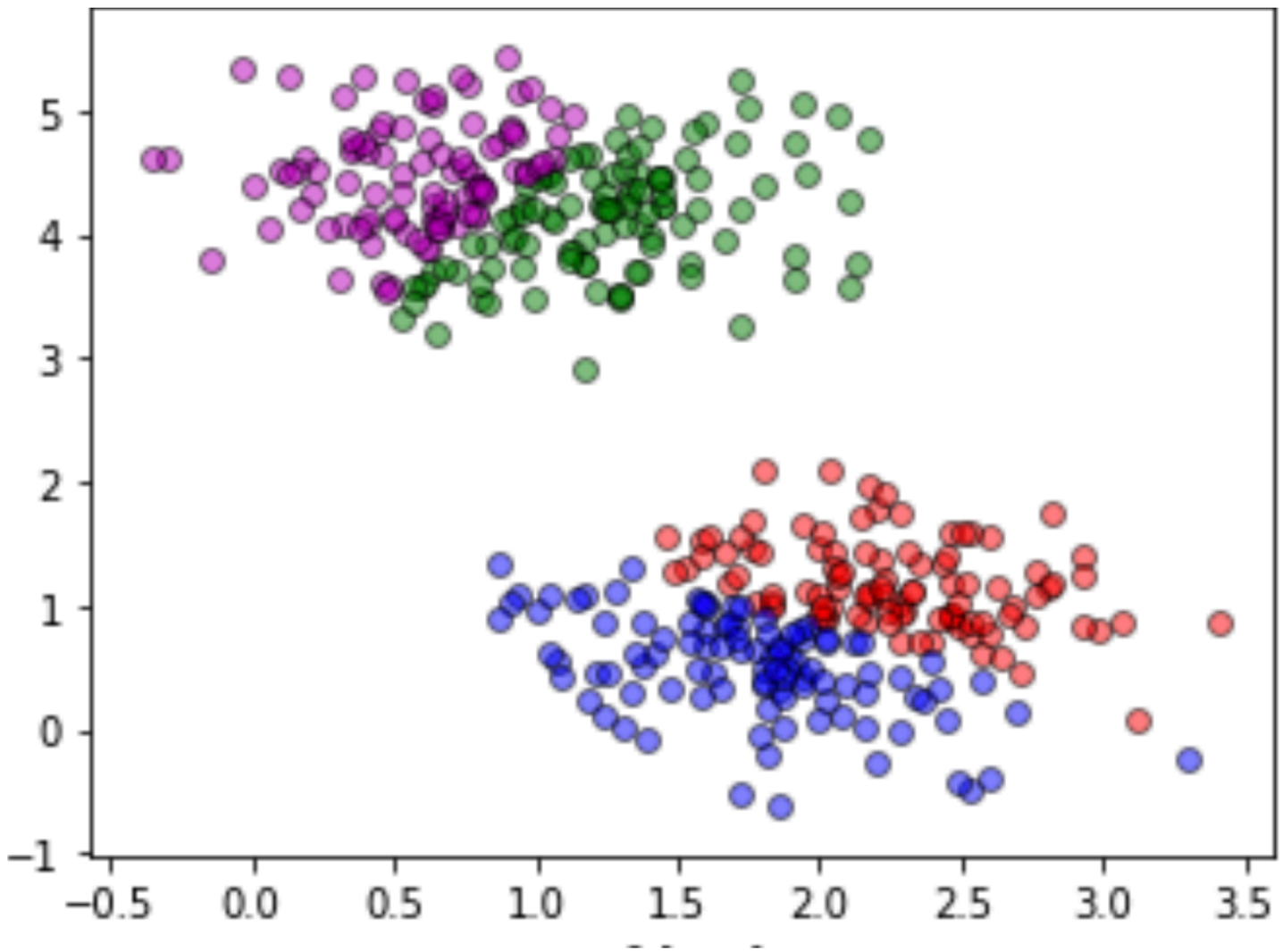
Clustering: example



Clustering: example



Clustering: example



Clustering challenges

- How to determine number of clusters?
- Is there any technique in which we do not need to choose the number of clusters?
- How can we assess the quality of clustering?
    - In classification/regression it is easy, we have labels/values. Here we don't.
    - There are some metrics, each of which targeting different things:
        - Davies-Bouldin Index (DBI),
        - Silhouette coefficient,
        - Dunn Index (DI),
        - etc.

Kmeans

- Very old (from 50s), yet very efficient and intuitive technique.
- Kmeans —> how to assign n examples into k clusters? NP-hard.
- It tries to decrease the variance within each cluster.

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \left\| \mathbf{x} - \boldsymbol{\mu}_i \right\|^2$$

#clusters

cluster center

i-th clusters

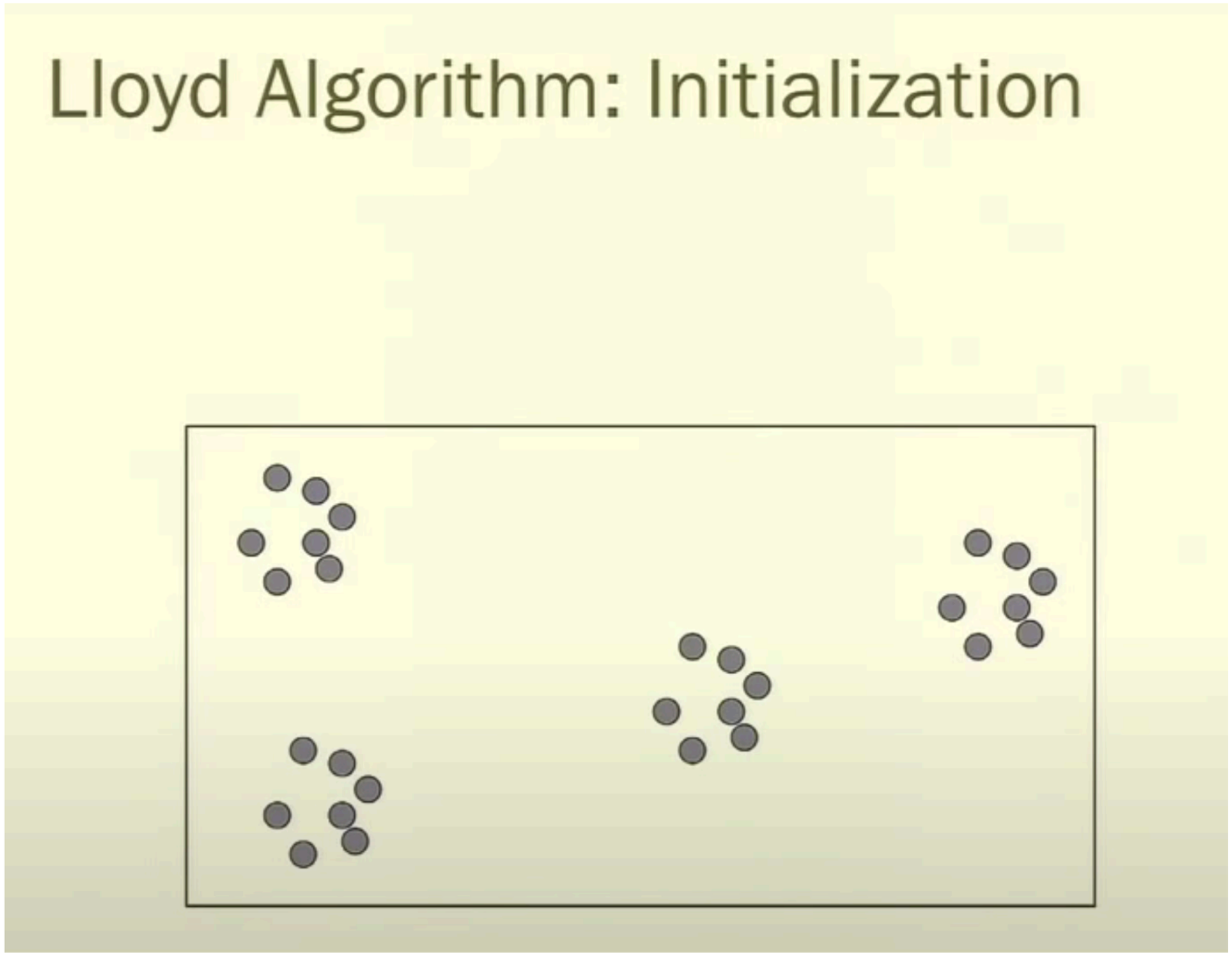
Naive Kmeans

- The most straightforward algorithm for Kmeans.
    - Start with k random instances as centers of clusters.
    - For each instance:
        - Assign it to the closest center.
    - For each cluster:
        - Recalculate the new center by averaging all the elements of the cluster.
    - Repeat until no change is observed in the cluster centers.
- How to choose the initial cluster centers? Kmeans is very very sensitive to that!
    - Randomly.
    - Random partition.

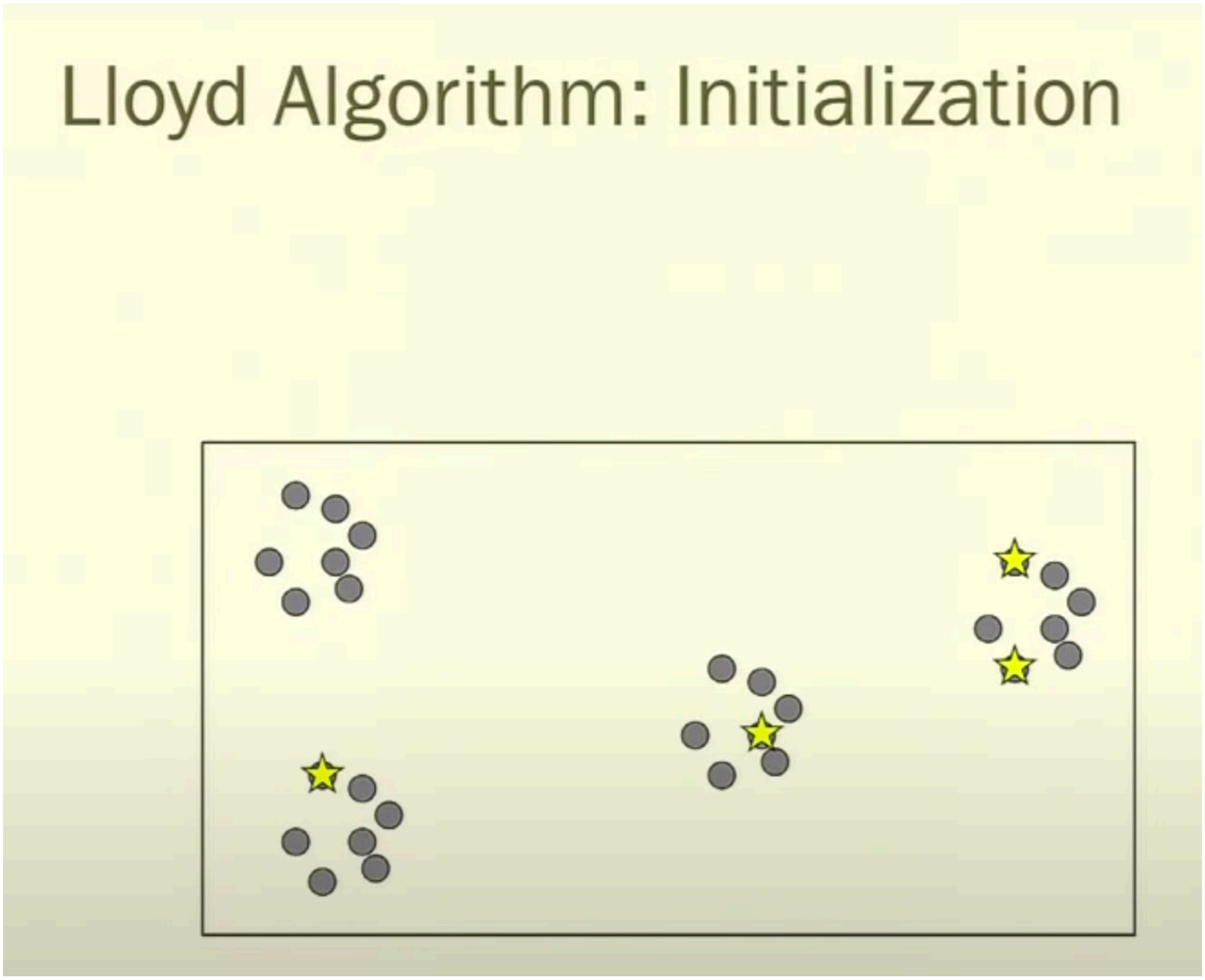
Naive Kmeans

- The most straightforward algorithm for Kmeans.
    - Start with k random instances as centers of clusters.
    - For each instance:
        - Assign it to the closest center.
    - For each cluster:
        - Recalculate the new center by averaging all the elements of the cluster.
    - Repeat until no change is observed in the cluster centers.
- How to choose the initial cluster centers? Kmeans is very very sensitive to that!
    - Randomly (Llyod Algorithm) —> what if you choose them all in the same sub-space?
    - Random partition.

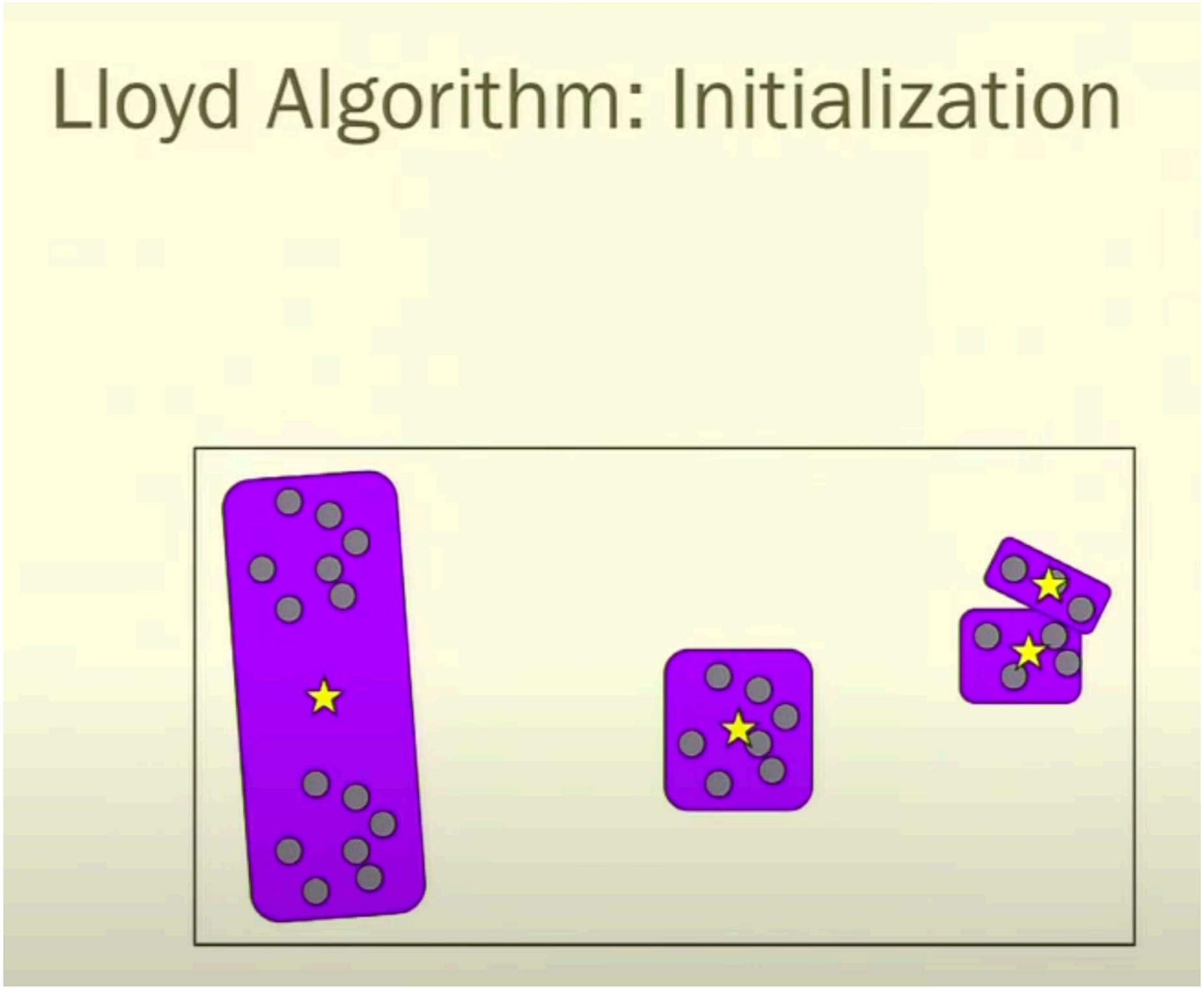
Kmeans: illustration



Kmeans: illustration



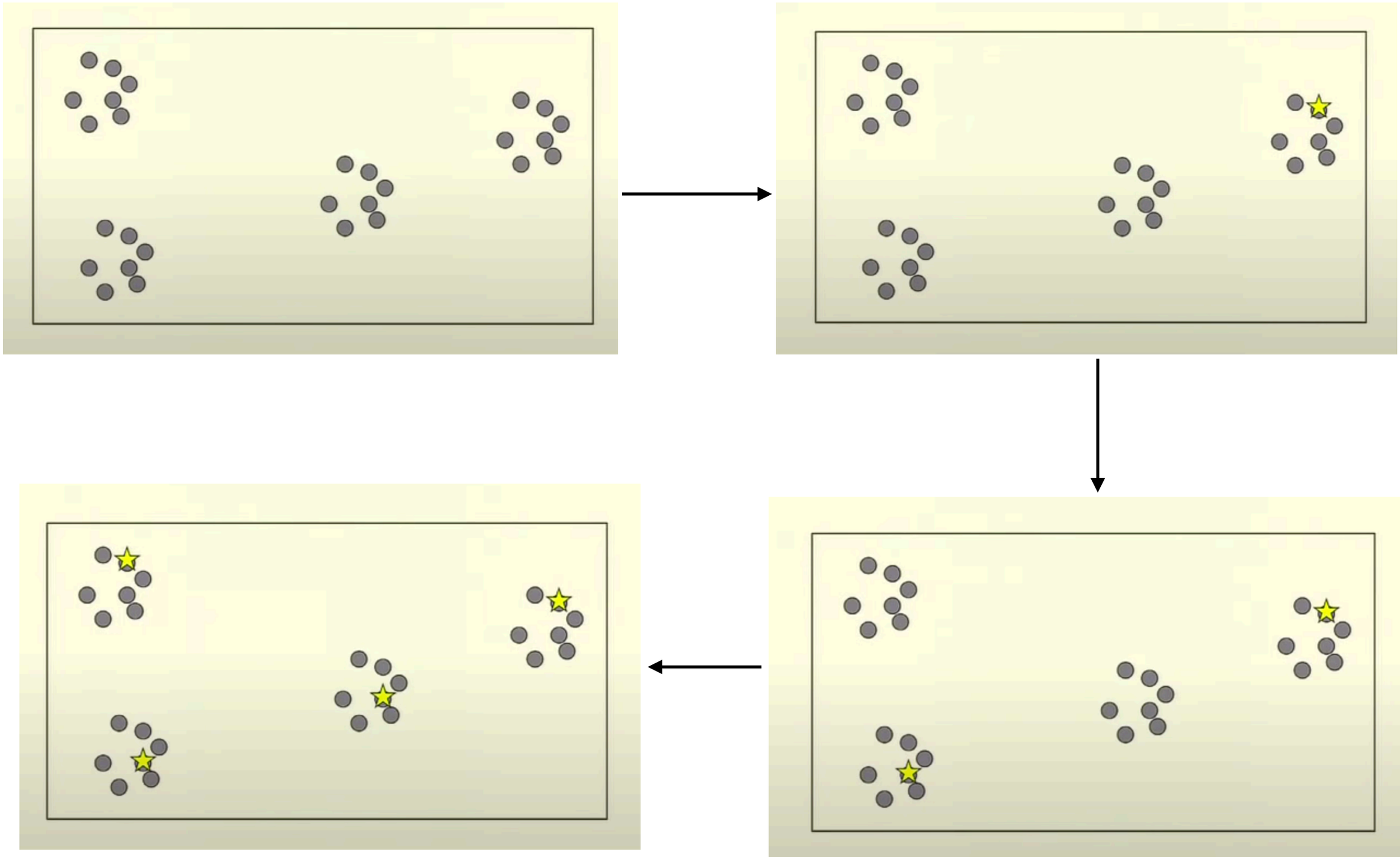
Kmeans: illustration



Kmeans++

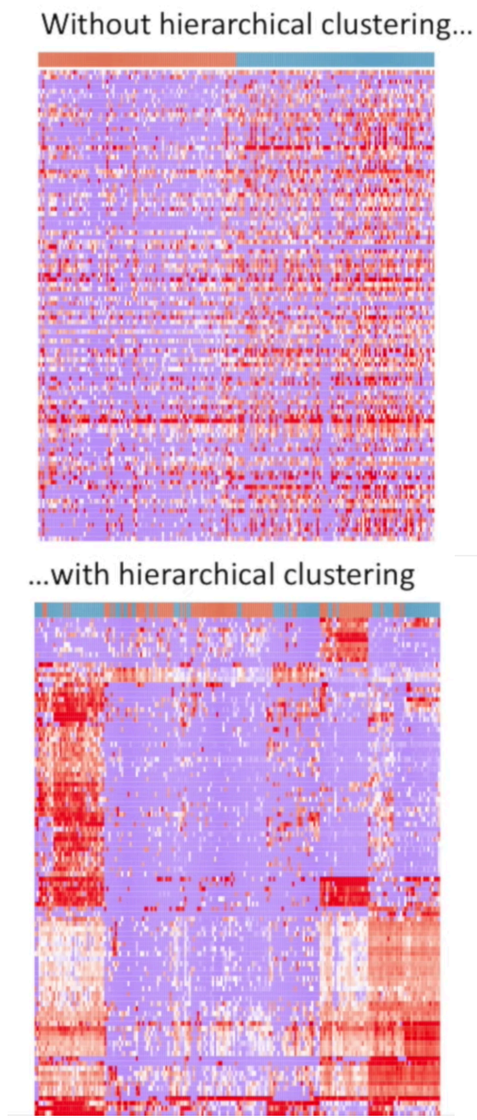
- Kmeans is very sensitive to initialization.
- Kmeans++ is technique to make Kmeans more stable and less dependent of the initialization.
- How does that work?
    - Choose the first center randomly.
        - For each instance, compute the distance w.r.t. the already chosen centers (at the beginning it is only one center).
        - Choose the second center using the weighted probability distribution proportional to the previous step.
        - Repeat this until k centers are chosen.
    - Proceed with normal Kmeans algorithm.
- How can it help? Is it deterministic? If not, how can we make it deterministic?

Kmeans++

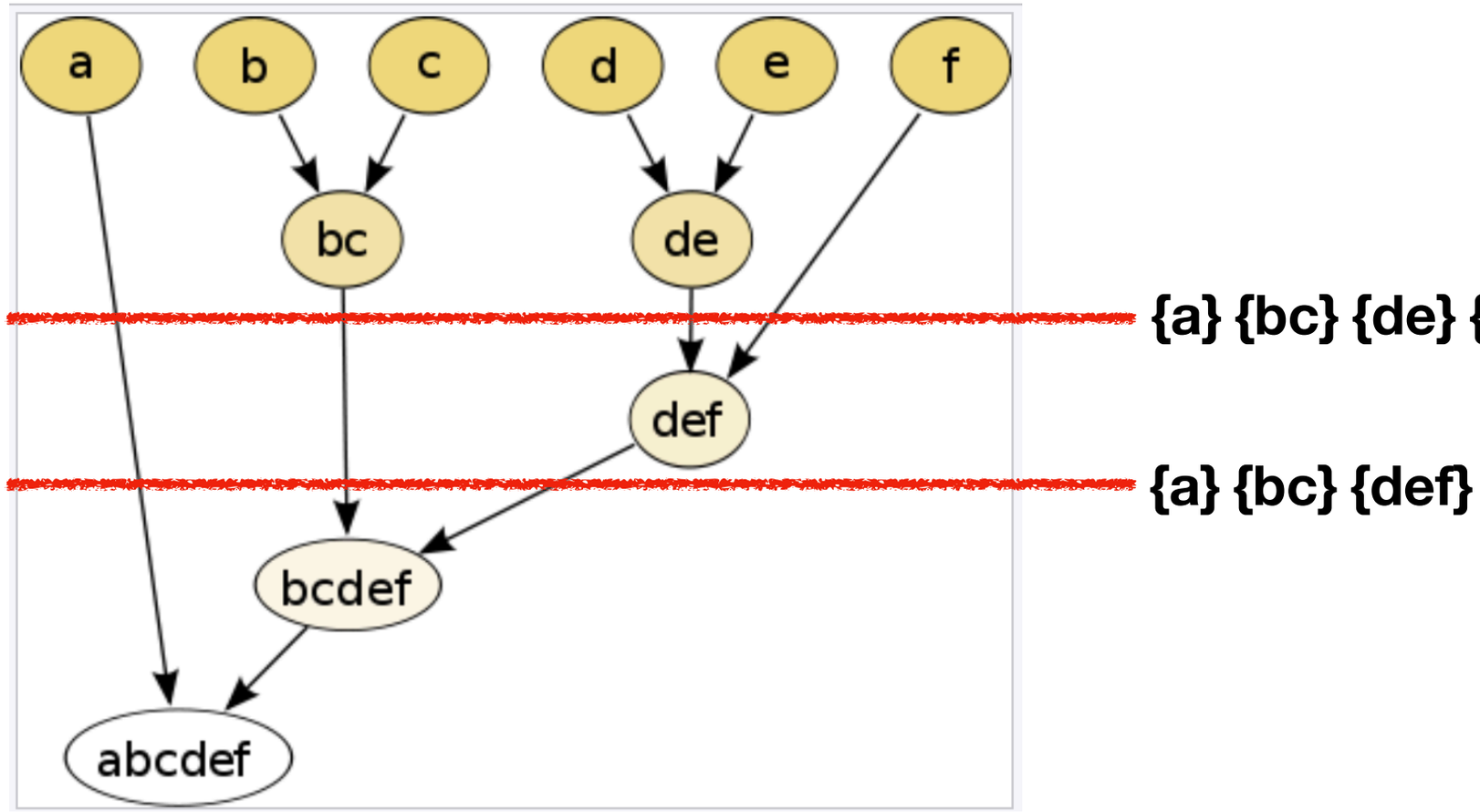


Hierarchical Clustering

- Another method to cluster the data.
- Traditionally used with the heat maps and gene expressions.
- Reorder the data to see which examples are similar ———————>
- Two approaches:
    - Agglomerative: a bottom-up approach —> 1 example, 1 cluster.
    - Divisive: top-down approach —> entire data = one cluster.
- Algorithm:
    1. Find the two closest instances.
    2. Merge them as a cluster and consider them as one instance.
    3. Go back to step 1 and continue until everything is merged.



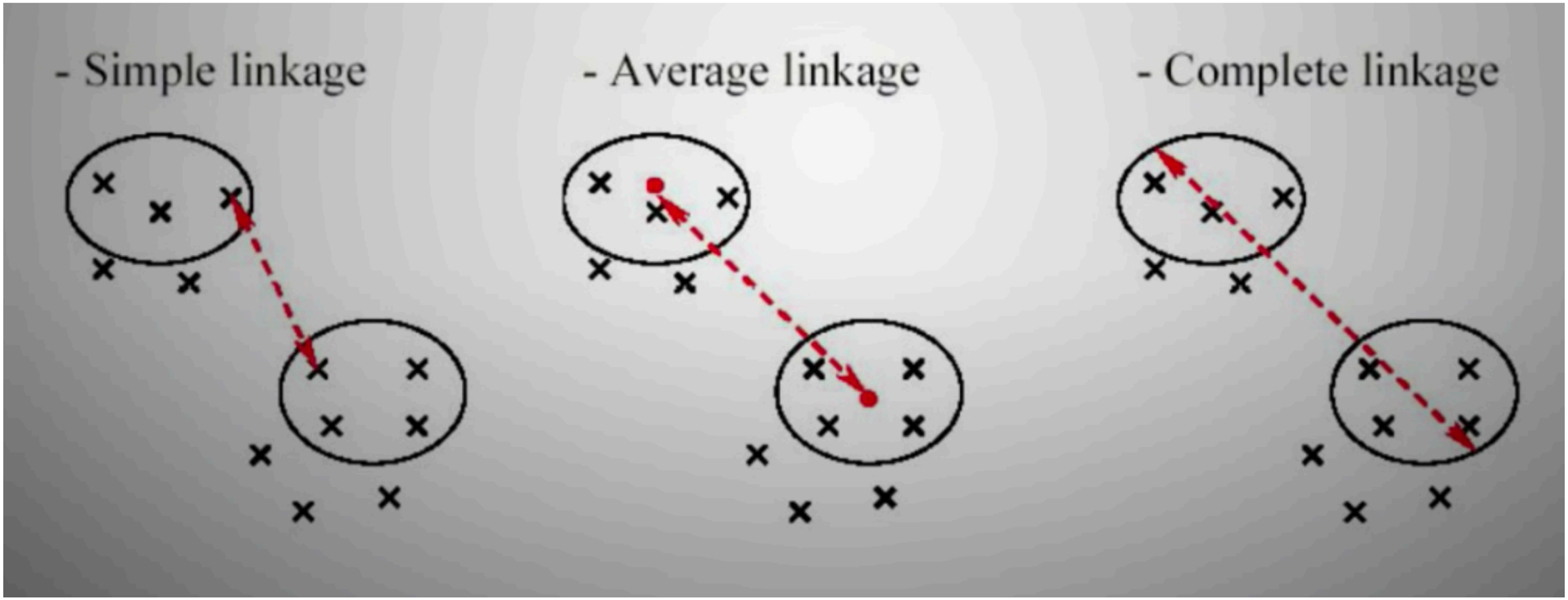
Agglomerative Clusters



- Build the tree —> cut based on level, cut based on minimum distance.
- Alternatively, keep merging until the desired number of clusters is reached.
- Distance measures: Euclidean, Manhattan, etc.

Different linkage approaches

- How to compete one instance or one cluster with another cluster:
    - Compare the centres (average)
    - Compare the further points (complete)
    - Compare the closest points (simple)
- The choice of linkage depends on the application and data.



Hierarchical clustering vs Kmeans

- Unlike Kmeans, hierarchical clustering cannot handle huge data (computationally expensive).
- Kmeans is sensitive to initialisation while hierarchical clustering is reproducible.
- Kmeans works very well with hyper spherical data.
- Kmeans needs the #clusters while with hierarchical clustering we can avoid that (completely?).

```
>>> from sklearn.cluster import AgglomerativeClustering
>>> import numpy as np
>>> X = np.array([[1, 2], [1, 4], [1, 0],
...               [4, 2], [4, 4], [4, 0]])
>>> clustering = AgglomerativeClustering().fit(X)
>>> clustering
AgglomerativeClustering()
>>> clustering.labels_
array([1, 1, 1, 0, 0, 0])
```

Clustering metrics

Silhouette coefficient

- Provides one score (between -1 and +1) per instance: how well each instance s clustered.
- How similar each object is with respect to its now cluster compared to other clusters.
- The silhouette score for the clustering is the average of the scores of all examples.

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i,j)$$ ⟶ The average of distances wrt all examples of the same cluster.

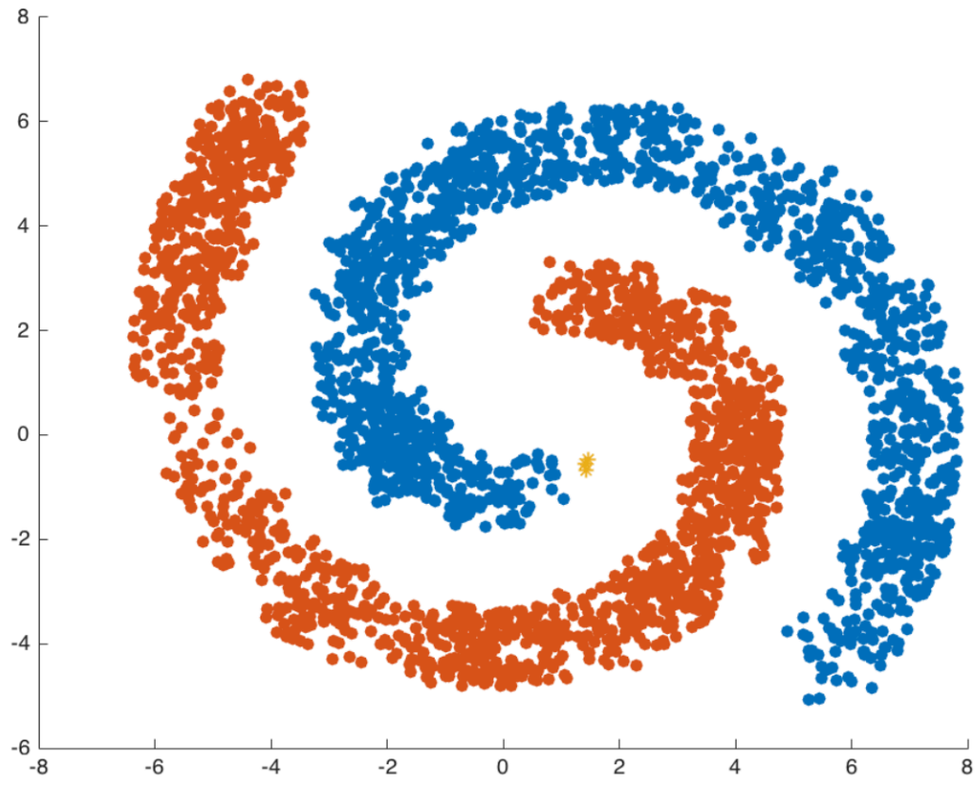
$$b(i) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i,j)$$ ⟶ Distance to the closest example from a *neighbouring* cluster.

$$s(i) = \begin{cases} 1 - a(i)/b(i), & \text{if } a(i) < b(i) \\ 0, & \text{if } a(i) = b(i) \\ b(i)/a(i) - 1, & \text{if } a(i) > b(i) \end{cases}$$ ⟶ The proportion between those two.

`sklearn.metrics.`**`silhouette_score`**(*X, labels)*

Density-based clustering

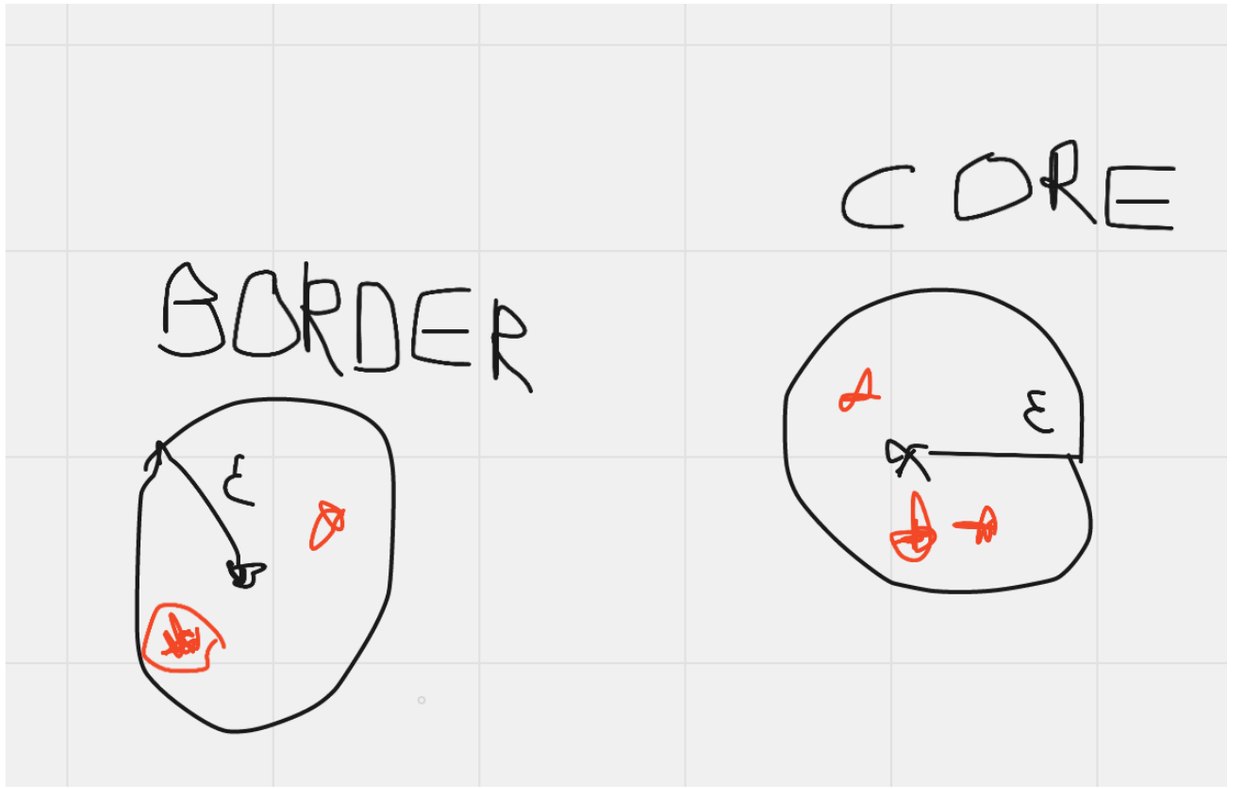
- So far —> centroid-based clustering and instance-based clustering.
- There is another type of clustering: density-based.
- It sees the data as groups of points, it naturally identifies the dense parts of the data.
- Intuitively, this is what human does by looking!
- Clusters —> dense areas, others are borders or outliers —> can *slightly* behave as outlier detection.



DBSCAN

Density-Based Spatial Clustering of Applications with Noise

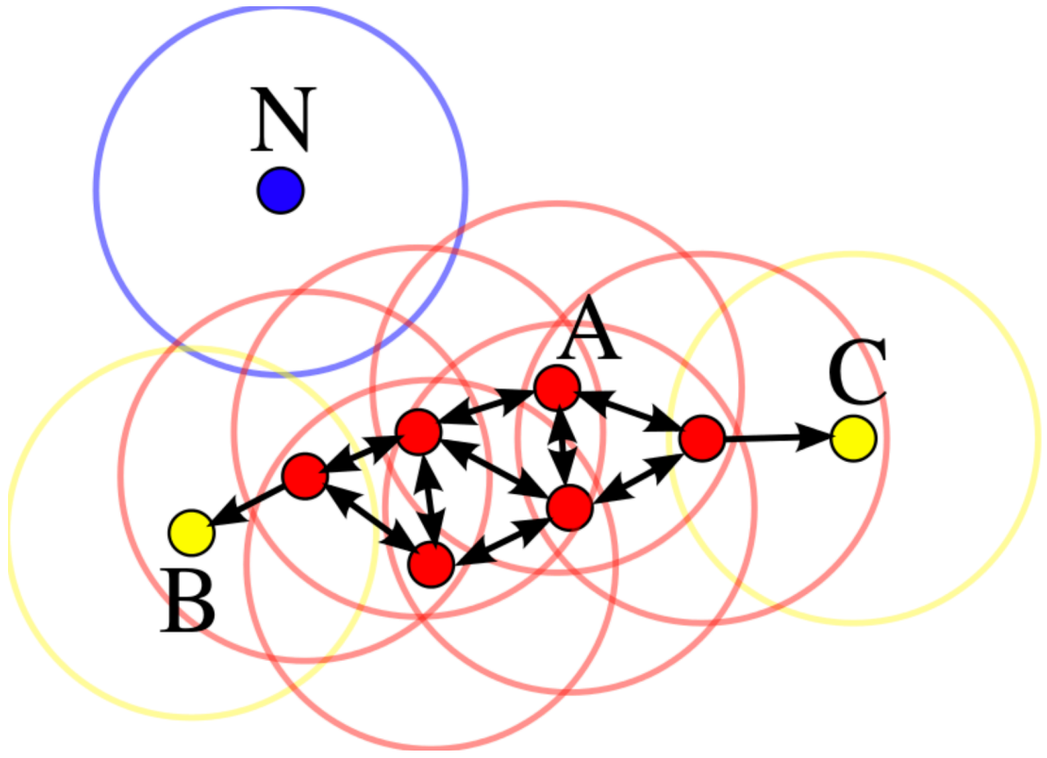
- The most know density-based clustering technique.
- Given a data point, it starts to grow clusters.
- Notions to understand in DBSCAN:
    - Core points,
    - Border points,
    - Outliers.
- Parameters of the method:
    - minPts —> minimum number of points.
    - epsilon: the radius to determine the boundary for each point
- Given a point:
    - Look at the epsilon-radius of the point: if there are at least minPts points (including itself) —> core point.
    - If not at least minPts points but at least another core point —> border point.
    - Otherwise —> outlier.



DBSCAN

Density-Based Spatial Clustering of Applications with Noise

- Pros:
    - Easily implemented.
    - Identifies coherent clusters.
    - Outlier detection mechanism.
    - No need for number of clusters.
    - Once the DIST matrix is computed, it is fast!
- Cons:
    - Works only for well shaped data.
    - Not entirely deterministic: depends on the order of processing.
    - Hard to find fixed number of clusters.
    - Tends to make clusters of the same size.
    - Hard to tune epsilon.



DBSCAN

Density-Based Spatial Clustering of Applications with Noise

```
DBSCAN(DB, distFunc, eps, minPts) {
    C = 0                                                   /* Cluster counter */
    for each point P in database DB {
        if label(P) ≠ undefined then continue               /* Previously processed in inner loop */
        Neighbors N = RangeQuery(DB, distFunc, P, eps)      /* Find neighbors */
        if |N| < minPts then {                              /* Density check */
            label(P) = Noise                                /* Label as Noise */
            continue
        }
        C = C + 1                                           /* next cluster label */
        label(P) = C                                        /* Label initial point */
        Seed set S = N \ {P}                                /* Neighbors to expand */
        for each point Q in S {                             /* Process every seed point */
            if label(Q) = Noise then label(Q) = C           /* Change Noise to border point */
            if label(Q) ≠ undefined then continue           /* Previously processed */
            label(Q) = C                                    /* Label neighbor */
            Neighbors N = RangeQuery(DB, distFunc, Q, eps) /* Find neighbors */
            if |N| ≥ minPts then {                          /* Density check */
                S = S U N                                   /* Add new neighbors to seed set */
            }
        }
    }
}
```

```
>>> from sklearn.cluster import DBSCAN
>>> import numpy as np
>>> X = np.array([[1, 2], [2, 2], [2, 3],
...               [8, 7], [8, 8], [25, 80]])
>>> clustering = DBSCAN(eps=3, min_samples=2).fit(X)
>>> clustering.labels_
array([ 0,  0,  0,  1,  1, -1])
>>> clustering
DBSCAN(eps=3, min_samples=2)
```

Semi-supervised learning

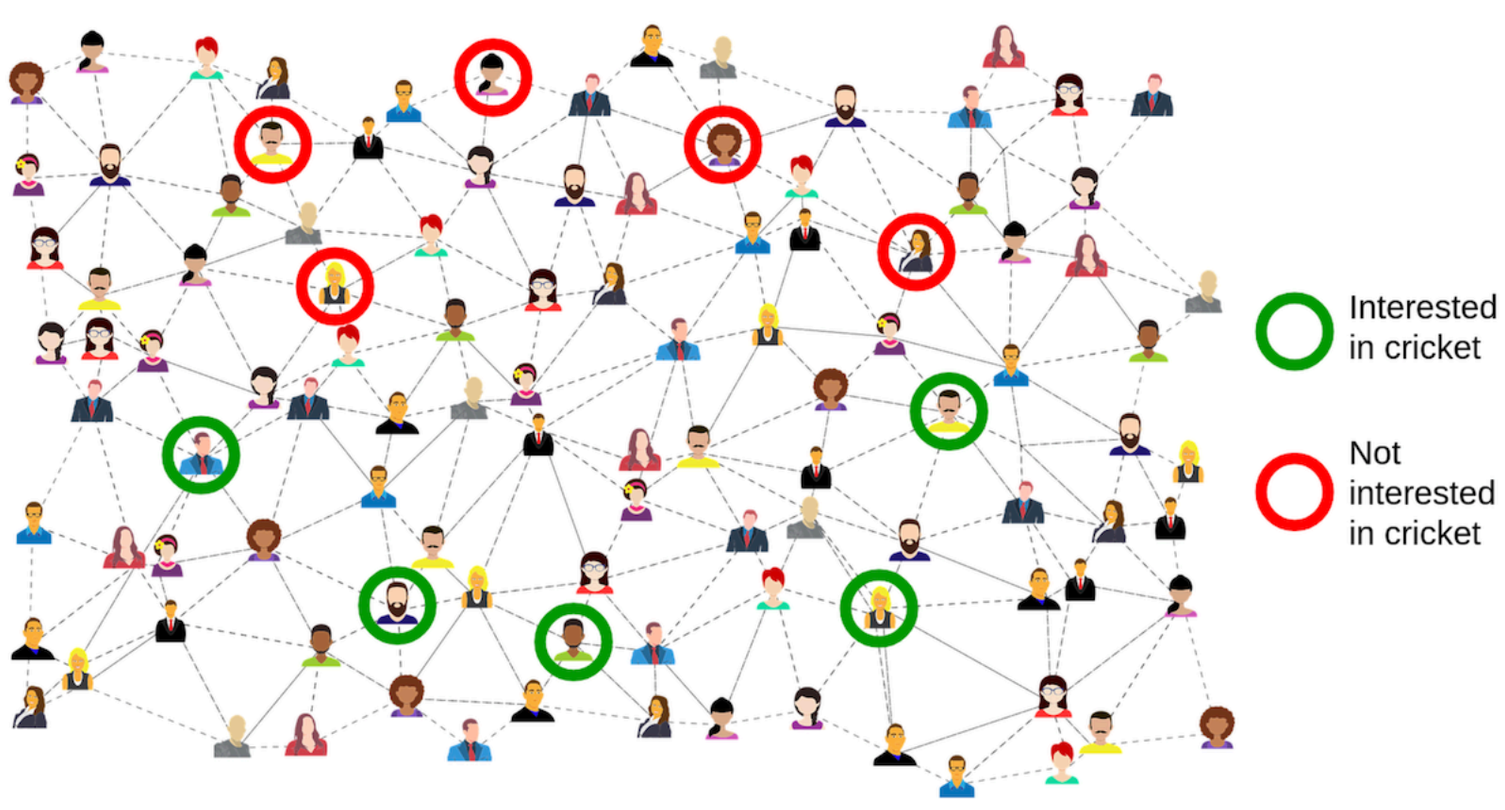
- Labelling data requires expensive human labor while unlabelled data is fairly easy to collect.
- Semi-supervised learning is all about learning good models based on few labeled data.
- Idea: learn a model based on labeled data, confidently pseudo-label the unlabelled data, relearn.
- Key points behind the semi-supervised learning [Zhou et al, 2004]:
    - Nearby points are likely to have the same labels (local).
    - Points in the same cluster are likely to have the same label (global).
- Two widely used techniques: label propagation and label spreading.

[Zhou et al, 2004]: Learning with Local and Global Consistency, NIPS

Semi-supervised learning

Label Propagation

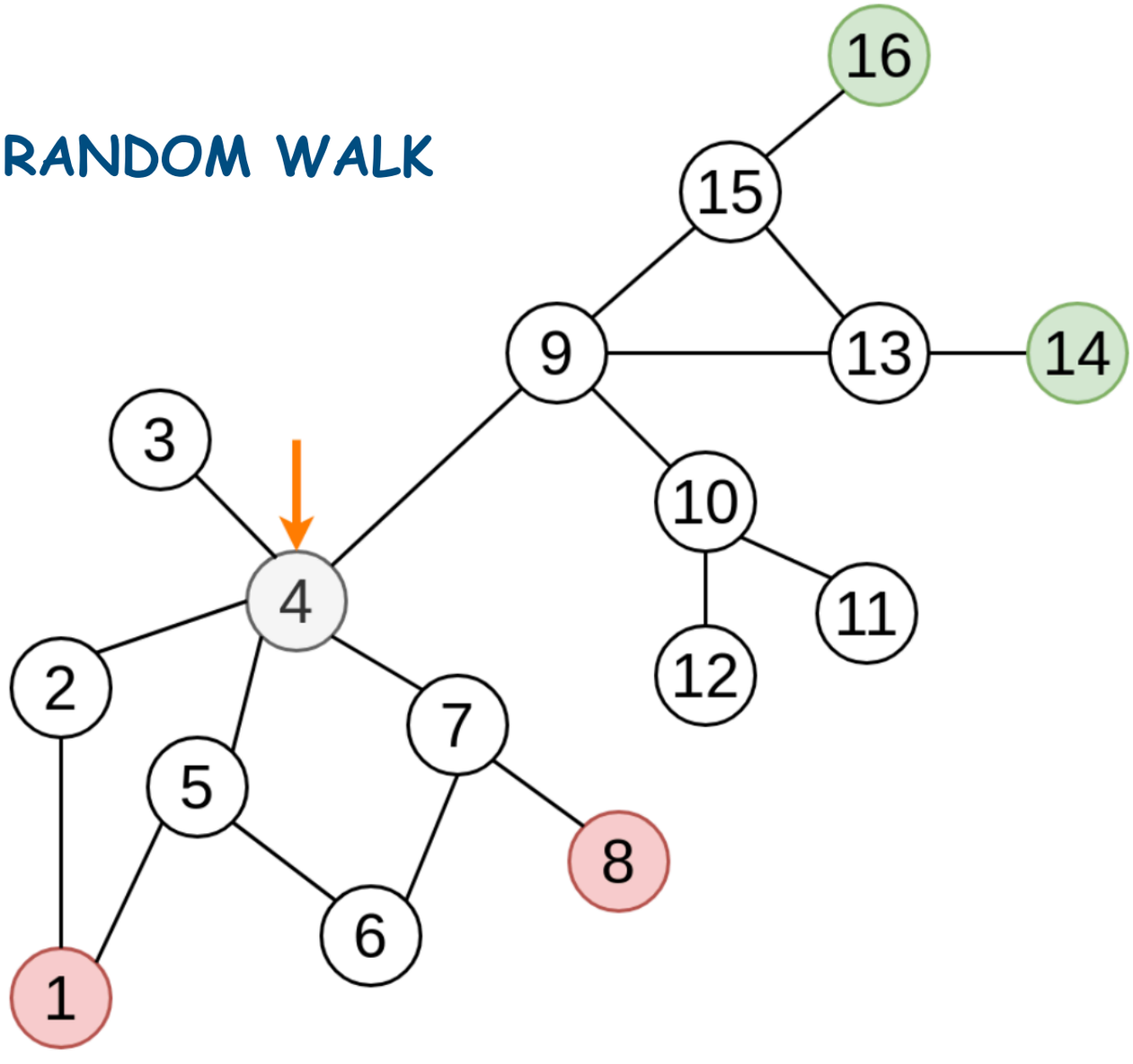
- Comes from the graphs —> the data needs to be modelled as a graph.
- How to label the unlabelled data given the connections between nodes.
- Intuitively: nodes close to reds —> red, and those close to greens —> green.



Semi-supervised learning

Label Propagation: how to do that?

RANDOM WALK



From 4 to red

$4 \rightarrow 7 \rightarrow 8$

$4 \rightarrow 7 \rightarrow 6 \rightarrow 5 \rightarrow 1$

$4 \rightarrow 5 \rightarrow 1$

$4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8$

$4 \rightarrow 2 \rightarrow 1$

From 4 to green

$4 \rightarrow 9 \rightarrow 15 \rightarrow 16$

$4 \rightarrow 9 \rightarrow 13 \rightarrow 14$

$4 \rightarrow 9 \rightarrow 13 \rightarrow 15 \rightarrow 16$

$4 \rightarrow 9 \rightarrow 15 \rightarrow 13 \rightarrow 14$

Red wins in count. We can also consider weights!

WHAT IS THE PROBLEM WITH LABEL PROPAGATION?

Semi-supervised learning

Label Spreading

- Label propagation propagates the known labels, what if they are wrong?
- Wrong info will get propagated into the entire set.
- Label spreading is designed to solve this problem by correcting the already labeled data.
- Label prop. uses graph Laplacian while label spreading uses the normalised graph Laplacian.

sklearn.semi_supervised.LabelSpreading

class sklearn.semi_supervised.**LabelSpreading**(*kernel='rbf', *, gamma=20, n_neighbors=7, alpha=0.2, max_iter=30, tol=0.001, n_jobs=None*) [source]

sklearn.semi_supervised.LabelPropagation

class sklearn.semi_supervised.**LabelPropagation**(*kernel='rbf', *, gamma=20, n_neighbors=7, max_iter=1000, tol=0.001, n_jobs=None*) ¶ [source]

Supervised learning: k-NN

- One of the simplest supervised learning techniques (instance-based learning).
- The idea is straightforward: an instance is classified *similar* to its neighbours.
- k-NN classification: popularity vote among k nearest neighbours.
- k-NN regression: the predicted value is the average of k nearest neighbours.
- Pros: intuitive, simple and easy to implement, assumption-free, no training,
- Cons: hard to tune k, not good with imbalanced data, sensitive to outliers, slow.
- To make it faster, one can use the approximative neighbour search techniques.
- How to tune k: use a part of labeled data to examine different values of k.