Introduction to Data Mining

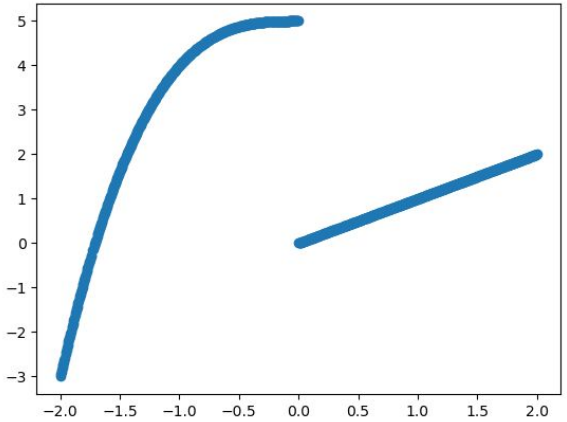
CS 145

Lecture 7:

Supervised: AdaBoost, KNN

Unsupervised: K-Means

Mixture Of Expert

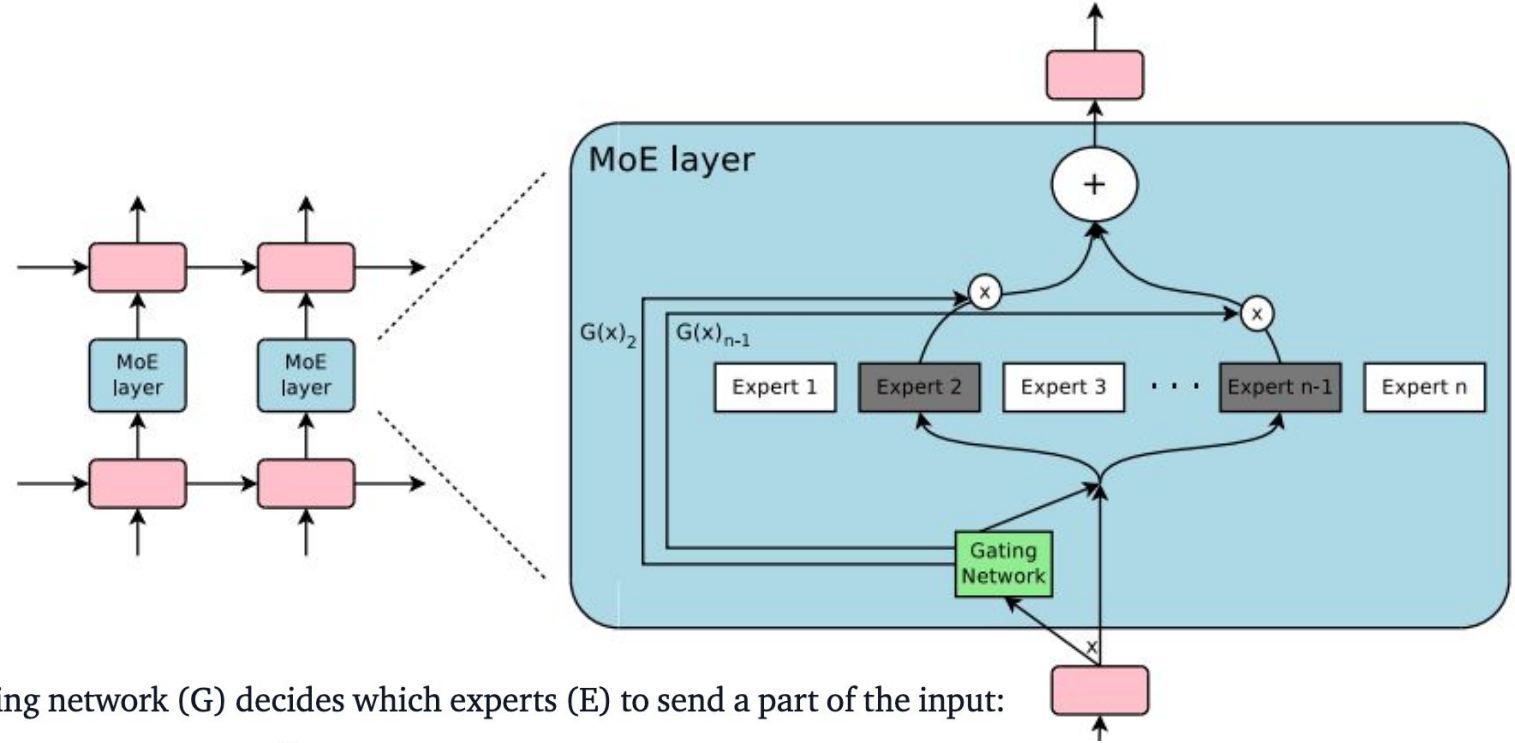


The key idea is to make each expert focus on predicting the right answer for the cases where it is already doing better than the other experts.

- This causes specialization.
- If we always average all the predictors, each model is trying to compensate for the combined error made by all the other models.

```
y = [x if x > 0 else (x**3 + 5)]
```

Mixture Of Expert



A learned gating network (G) decides which experts (E) to send a part of the input:

$$y = \sum_{i=1}^{n} G(x)_i E_i(x) \qquad G_\sigma(x) = \text{Softmax}(x \cdot W_g)$$

Boosting

“The Strength of Weak Classifiers”*

Terminology: Weak Learning

- **Error rate:** $\varepsilon_{h,P} = E_{P(x,y)}\left[1_{[h(x)\neq y]}\right]$
- **Weak Classifier:** $\varepsilon_{h,P}$ slightly better than 0.5
  - Slightly better than random guessing

Shallow Decision Trees are Weak Classifiers!

Weak Learners are Low Variance & High Bias!

How to "Boost" Weak Models?

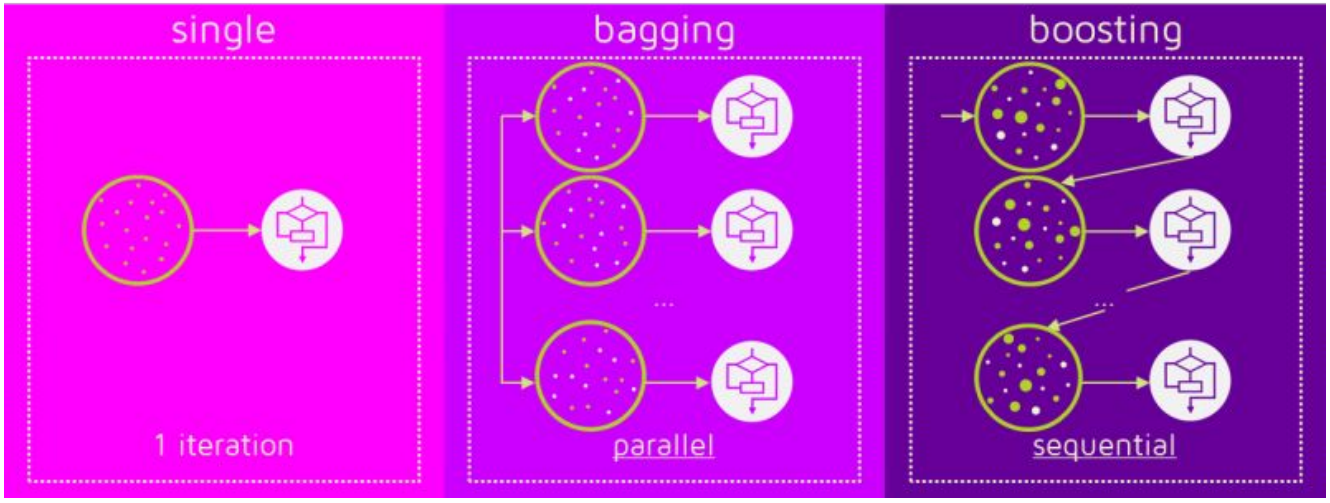
$$\underbrace{E_S\left[L_P(h_S)\right]}_{\text{Expected Test Error}} = E_{(x,y)\sim P(x,y)}\left[\underbrace{E_S\left[\left(h_S(x)-H(x)\right)^2\right]}_{\text{Variance Term}} + \underbrace{\left(H(x)-y\right)^2}_{\text{Bias Term}}\right]$$

Expected Test Error
Over randomness of S
(Squared Loss)

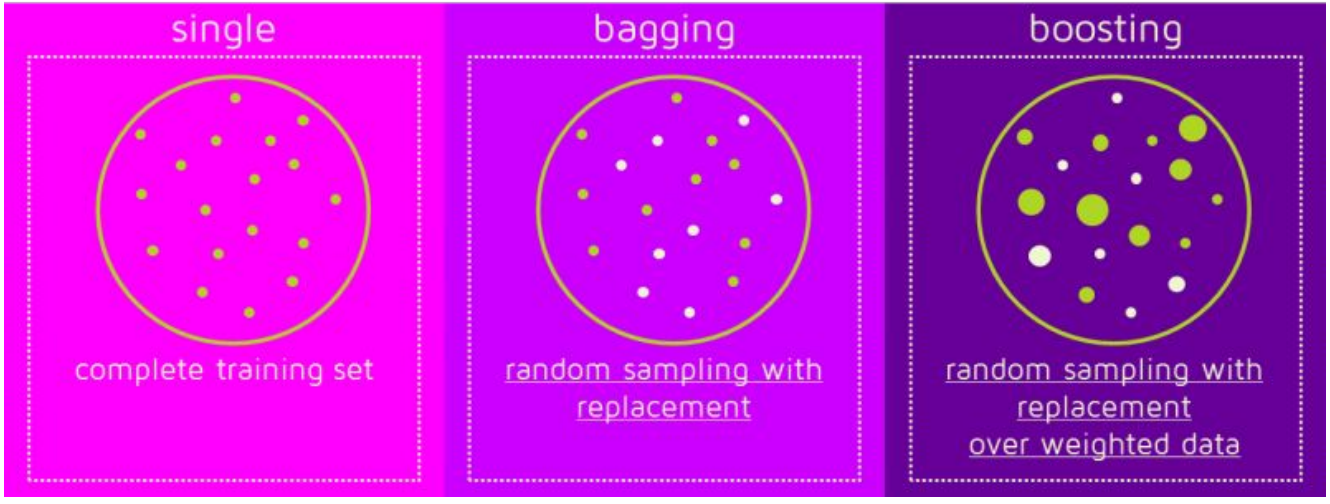
"Average prediction on x" → $H(x) = E_S\left[h_S(x)\right]$

- Weak Models are High Bias & Low Variance
- Bagging would not work
  - Reduces variance, not bias

Pipeline:



Data Generation:



https://www.kaggle.com/code/prashant111/bagging-vs-boosting

AdaBoost

Adaptive Boosting for Classification

Recap: AdaBoost

- Gradient Descent in Function Space
  - Space of weak classifiers
- Final model = linear combination of weak classifiers
  - $h(x) = \text{sign}(a_1 h_1(x) + \ldots + a_n h_n(x))$
  - I.e., a point in Function Space
- Iteratively creates new training sets via reweighting
  - Trains weak classifier on reweighted training set
  - Derived via minimizing residual Exp Loss

AdaBoost = Minimizing Exp Loss

- Init $D_1(x) = 1/N$
- Loop t = 1…n:
  - Train classifier $h_t(x)$ using (S,D_t)
  - Compute error on (S,D_t): $\varepsilon_t \equiv L_{D_t}(h_t) = \sum_i D_t(i) L(y_i, h_t(x_i))$
  - Define step size a_t: $a_t = \frac{1}{2}\log\left\{\frac{1-\varepsilon_t}{\varepsilon_t}\right\}$
  - Update Weighting: $D_{t+1}(i) = \frac{D_t(i)\exp\{-a_t y_i h_t(x_i)\}}{Z_t}$
- **Return:** $h(x) = \text{sign}(a_1 h_1(x) + \ldots + a_n h_n(x))$

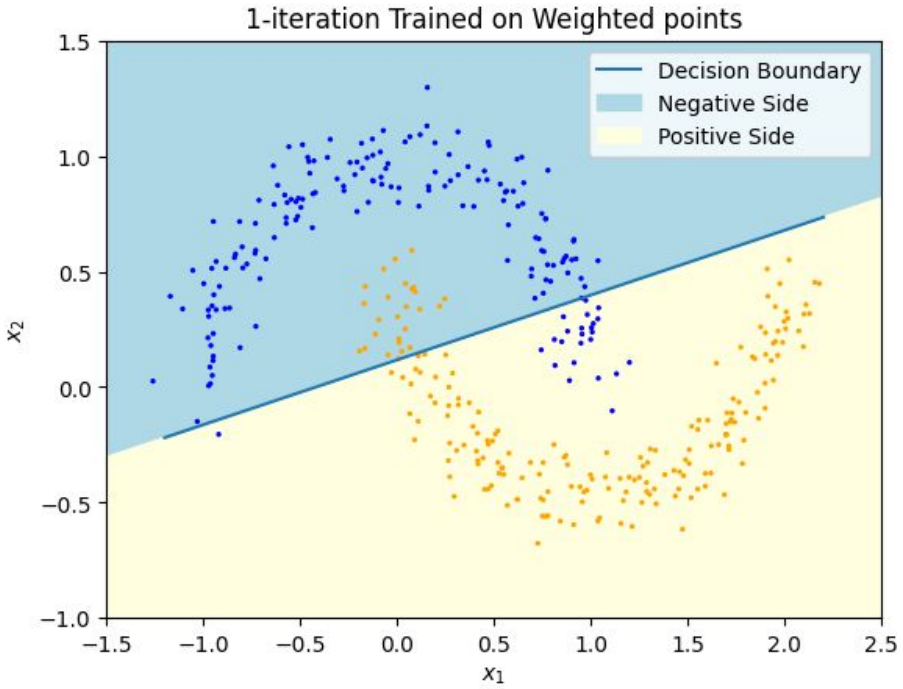
$$S = \{(x_i, y_i)\}_{i=1}^{N}$$

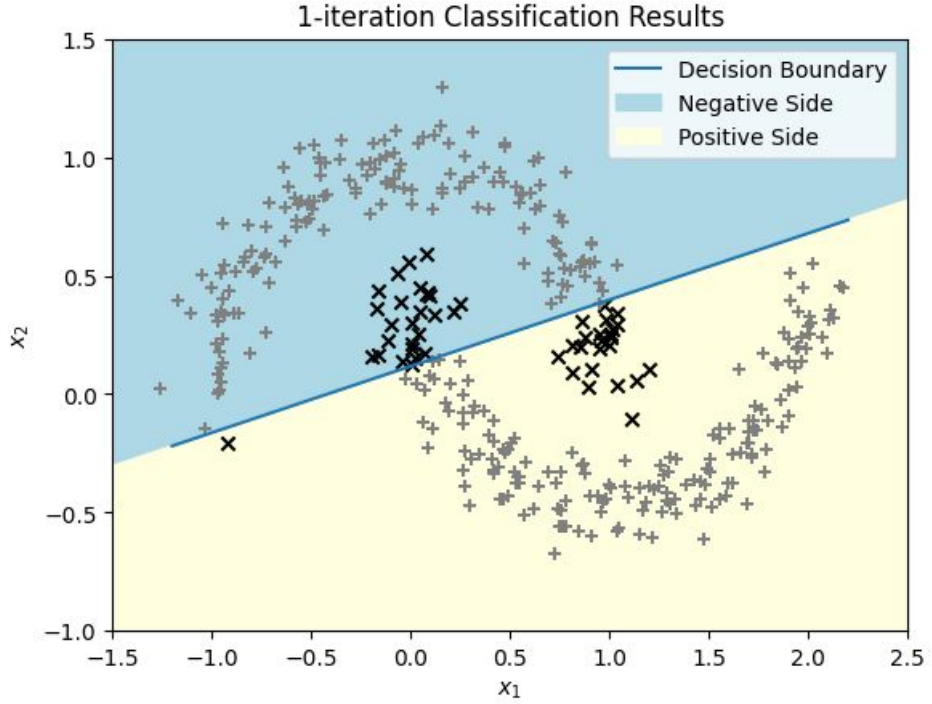
$$y_i \in \{-1, +1\}$$

Data points reweighted according to Exp Loss!

Normalization Factor s.t. D_{t+1} sums to 1.

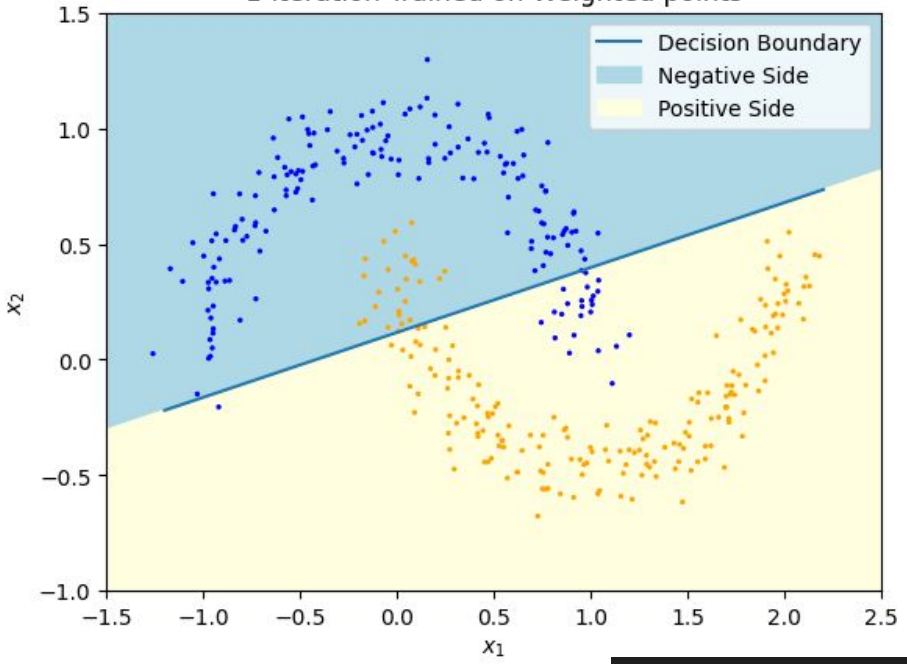
Using Linear Regression as Base

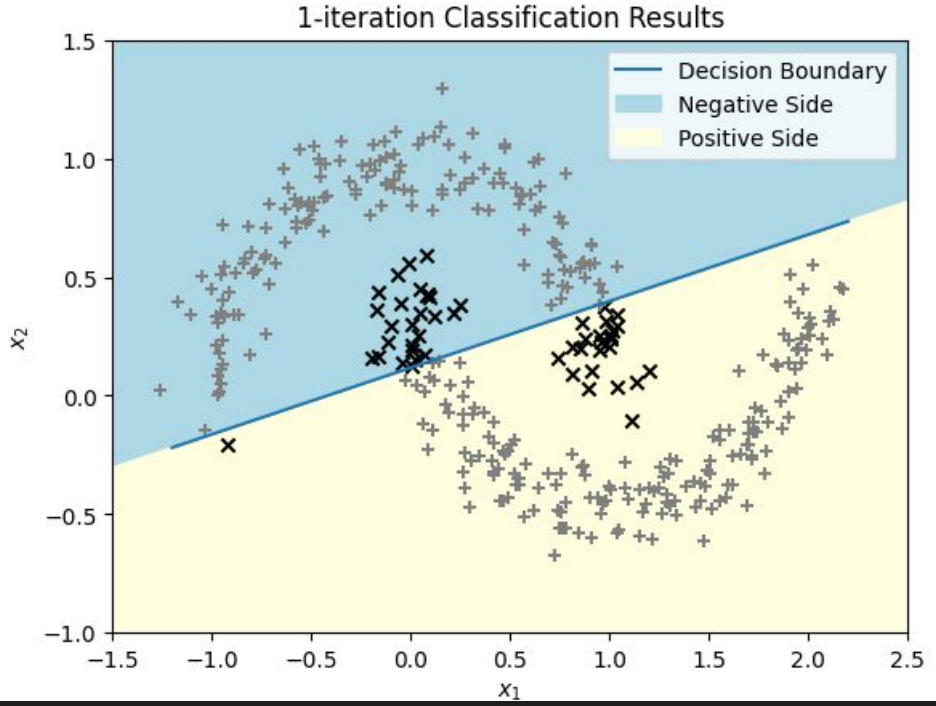




$$\varepsilon_t \equiv L_{D_t}(h_t) = \sum_i D_t(i) L(y_i, h_t(x_i))$$

```
# Compute weighted error rate use 0/1 Loss
incorrect = predictions != y_train
weighted_error = np.sum(instance_weights * incorrect)
```



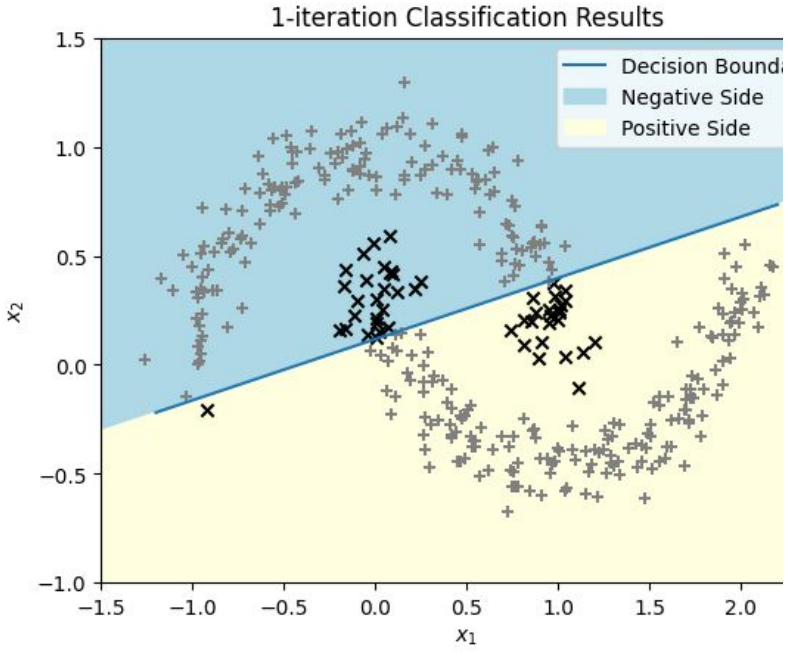


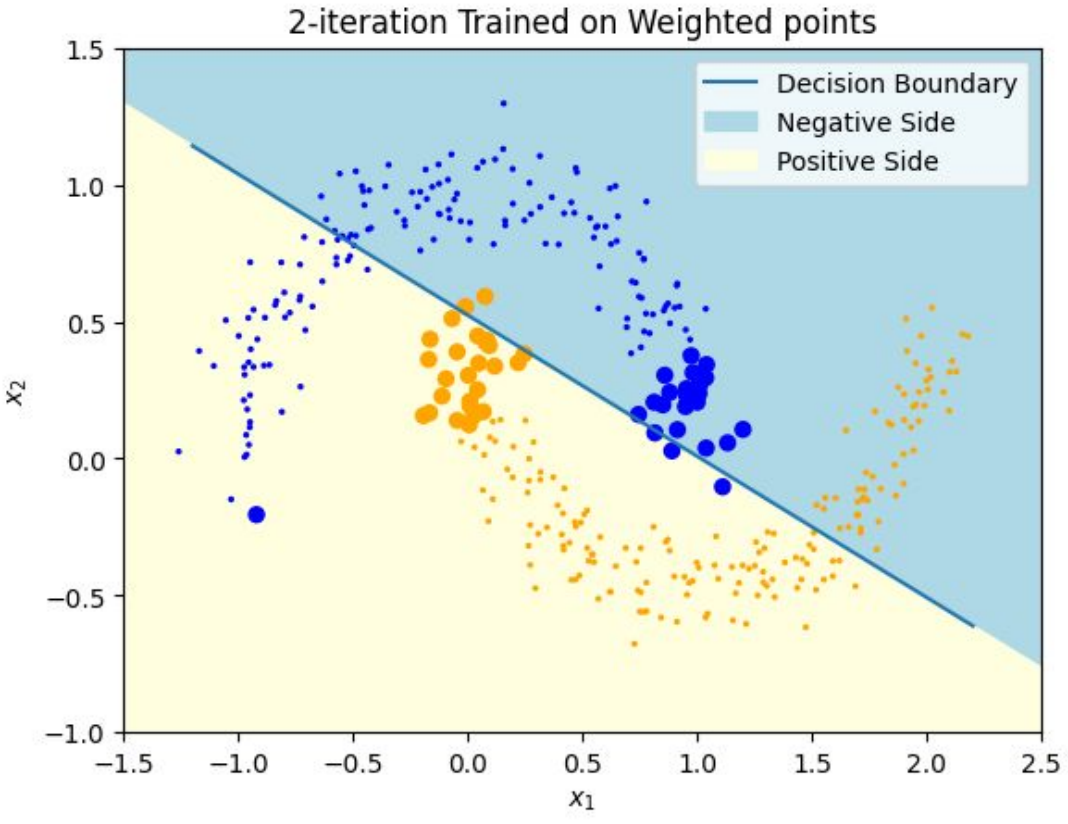
$$\varepsilon_t \equiv L_{D_t}(h_t) = \sum_i D_t(i) L(y_i, h_t(x_i))$$

step size a_t: $$a_t = \frac{1}{2}\log\left\{\frac{1-\varepsilon_t}{\varepsilon_t}\right\}$$

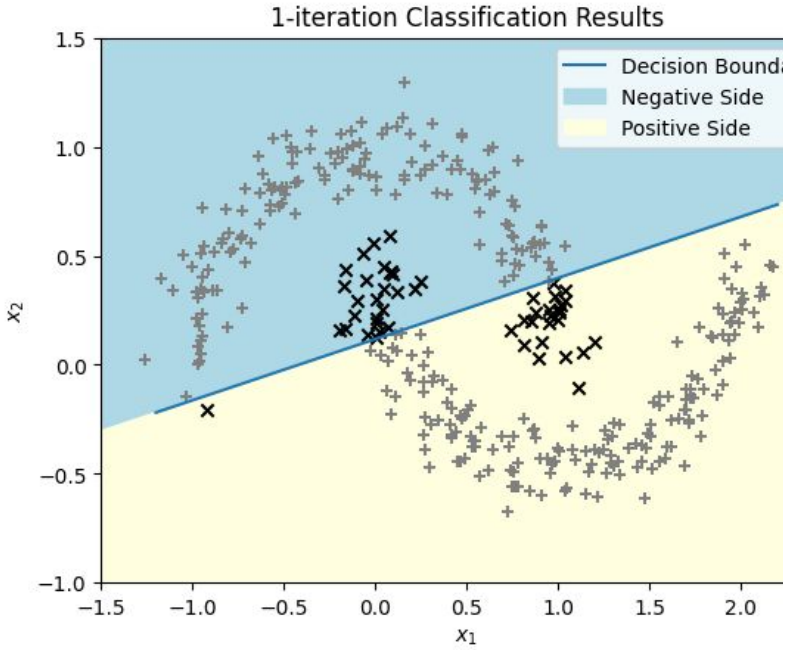
```
# Compute weighted error rate use 0/1 Loss
incorrect = predictions != y_train
weighted_error = np.sum(instance_weights * incorrect)
# Calculate the predictor's weight using its error
predictor_weight = np.log((1 - weighted_error) / (weighted_error))
```

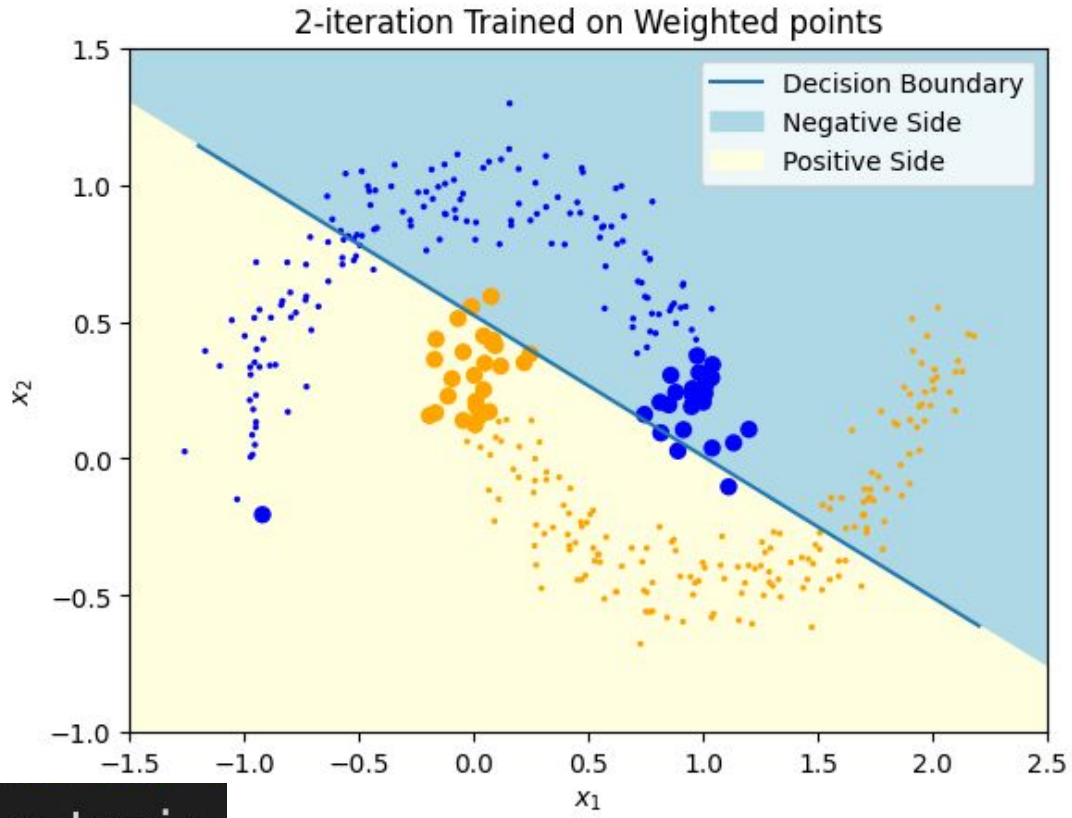
```
Weighted Error: 0.133,  Model Weights: 1.404
```



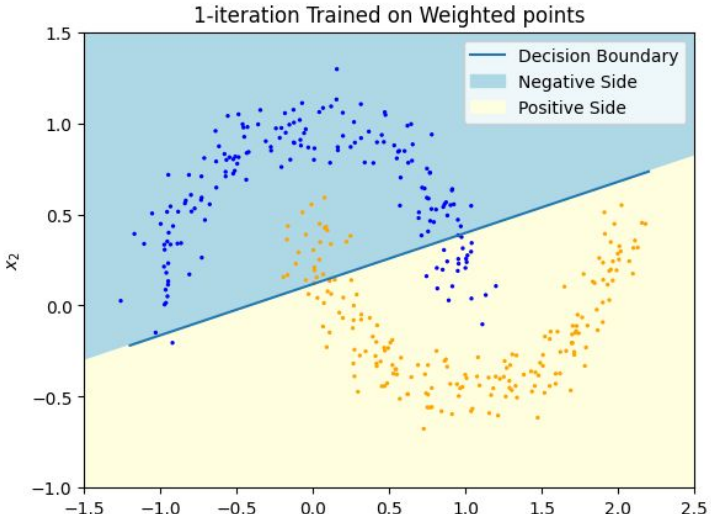


Update Weighting: $$D_{t+1}(i) = \frac{D_t(i)\exp\{-a_t y_i h_t(x_i)\}}{Z_t}$$

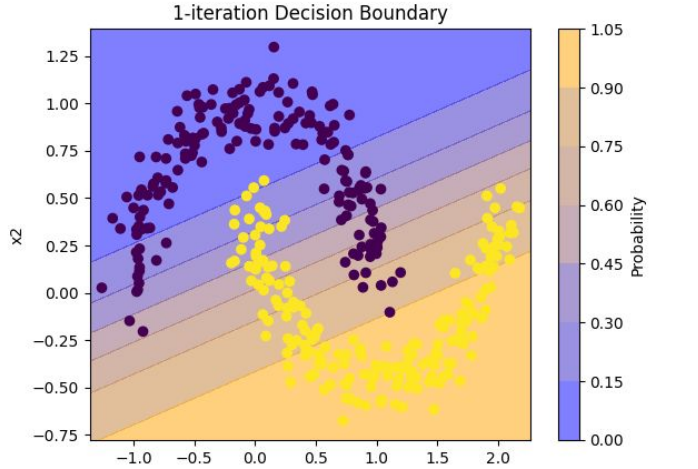


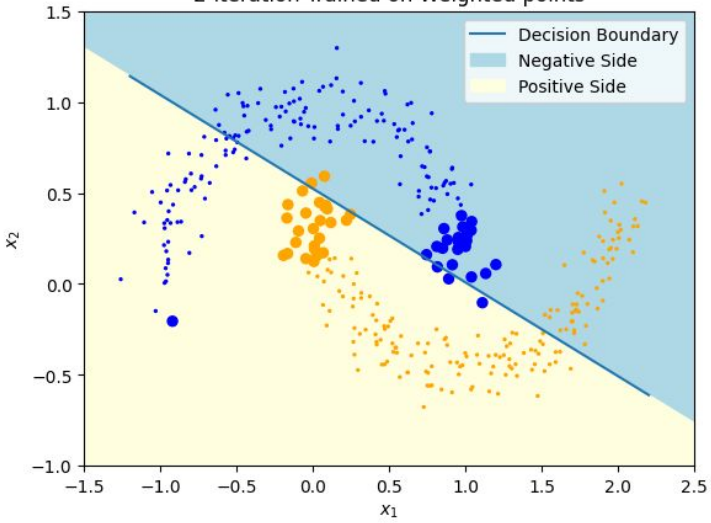



```
incorrect = predictions != y_train
# Update instance weights
instance_weights *= np.exp(predictor_weight * (incorrect * 2 - 1))
instance_weights = instance_weights / np.sum(instance_weights)
```



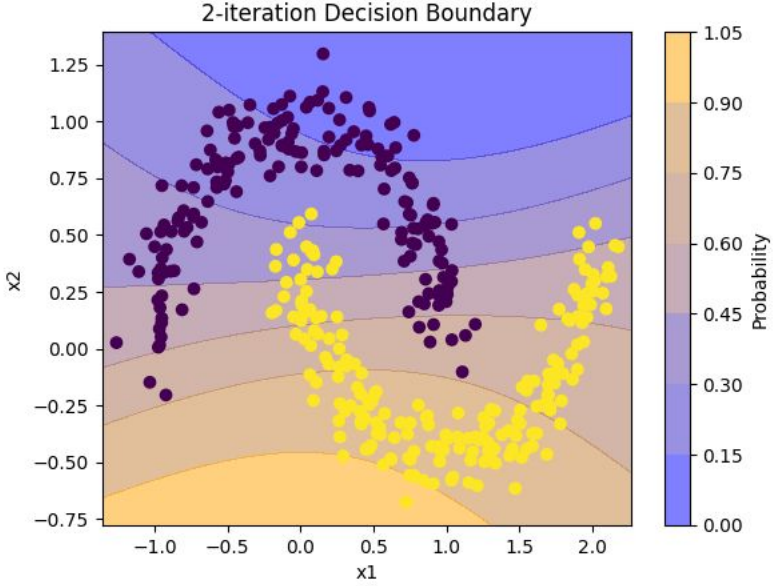
Model Weights: 1.404

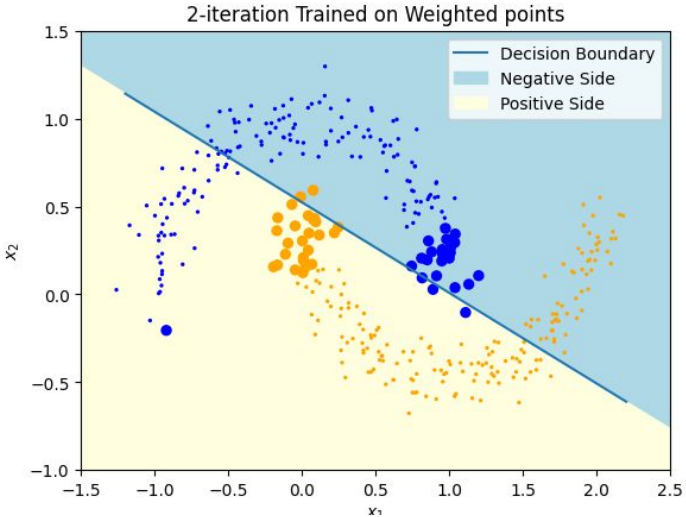


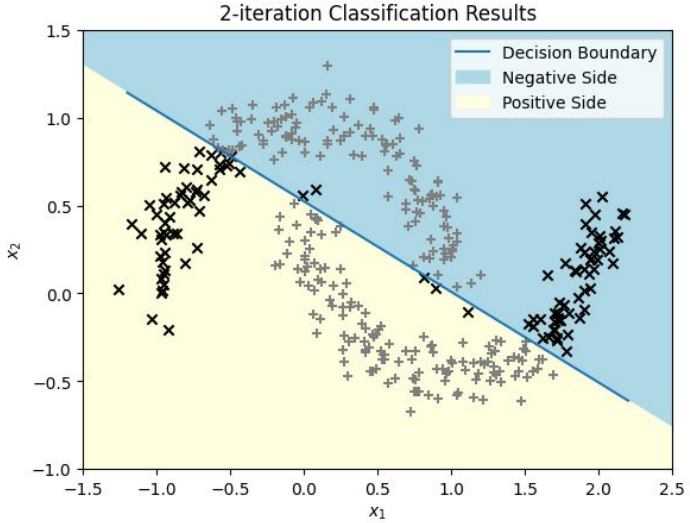


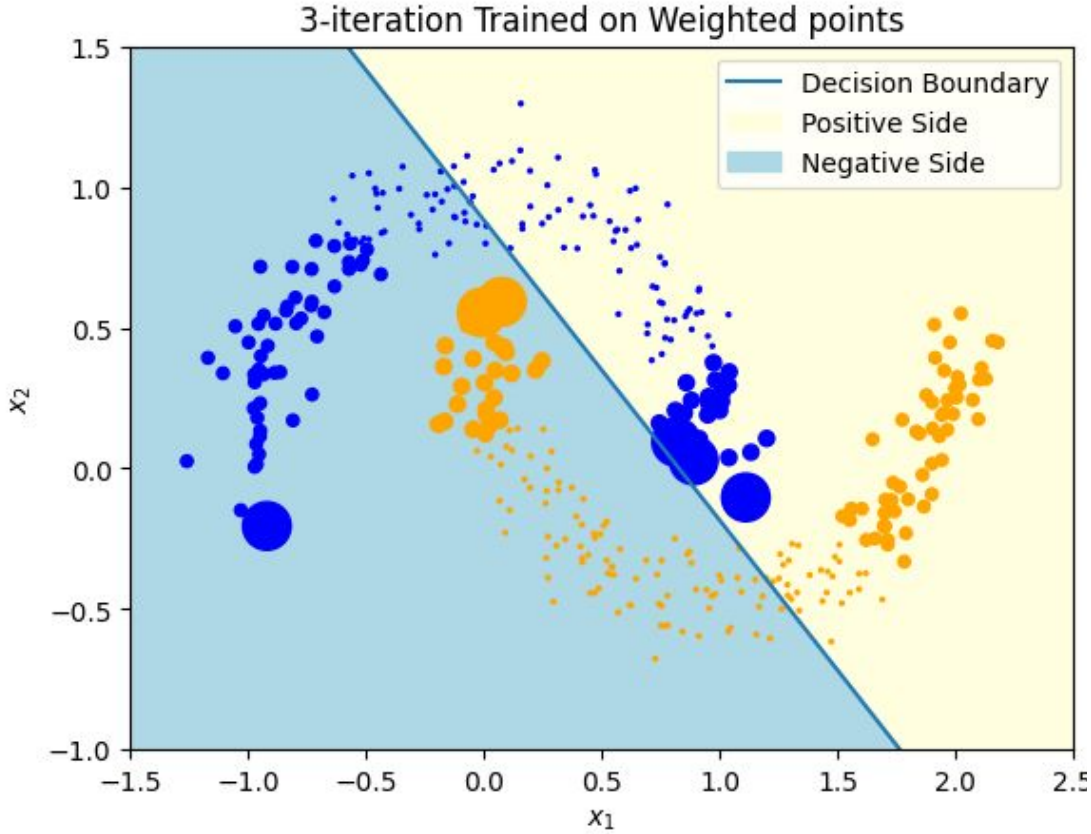
Model Weights: 1.147

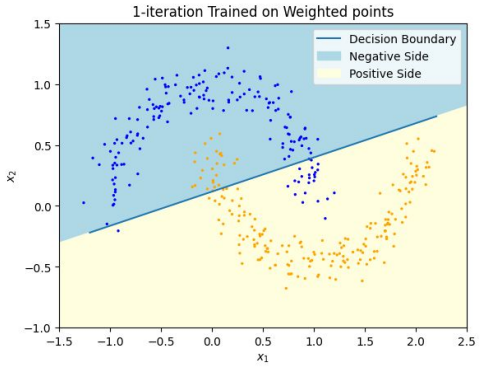
1.4 * linear-1 +
1.15 * linear-2

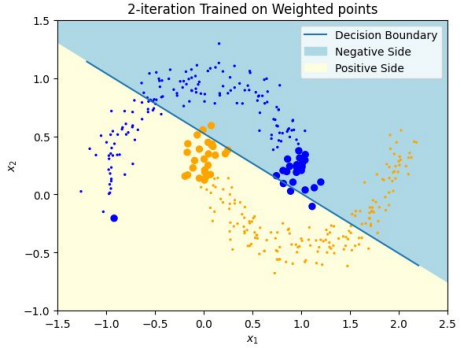


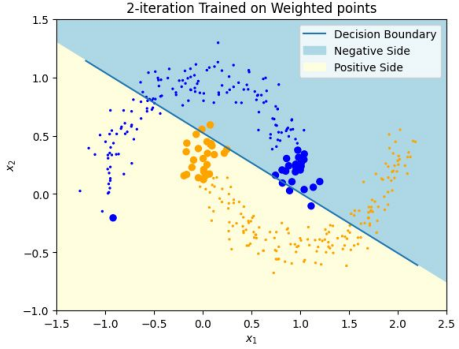

2-iteration Trained on Weighted points
Decision Boundary
Negative Side
Positive Side
x_1
x_2


2-iteration Classification Results
Decision Boundary
Negative Side
Positive Side
x_1
x_2


3-iteration Trained on Weighted points
Decision Boundary
Positive Side
Negative Side
x_1
x_2



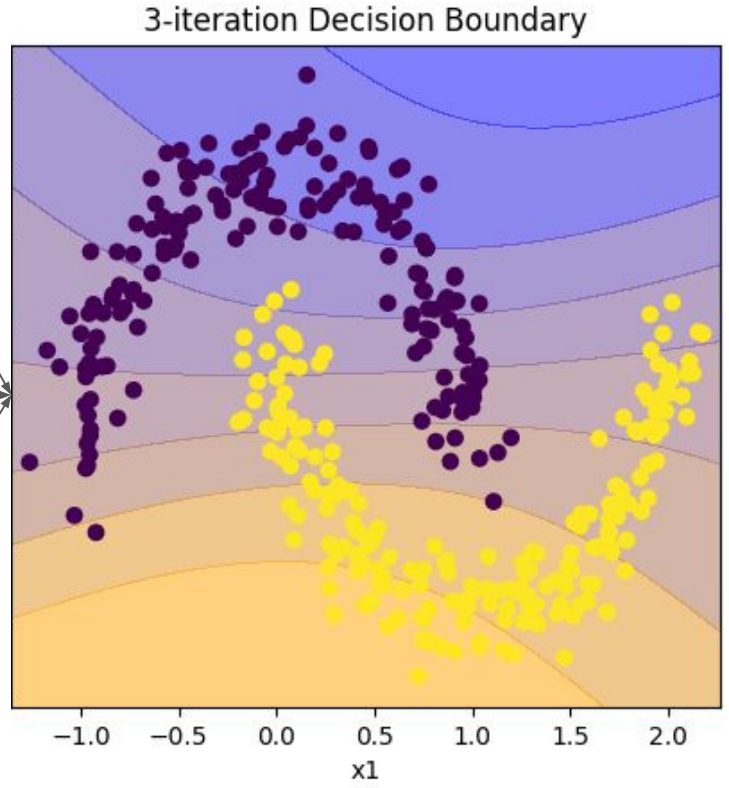


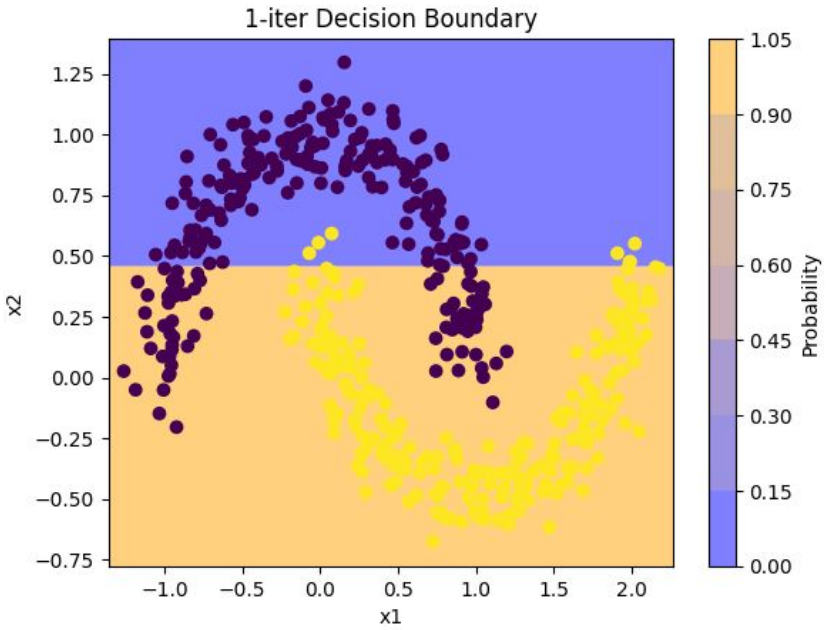


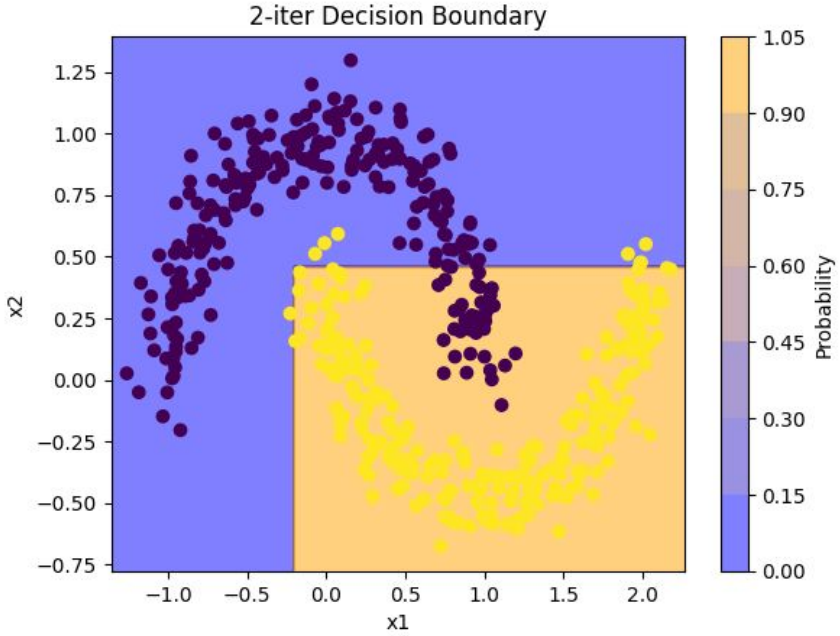

```
Model Weights: 1.404
```

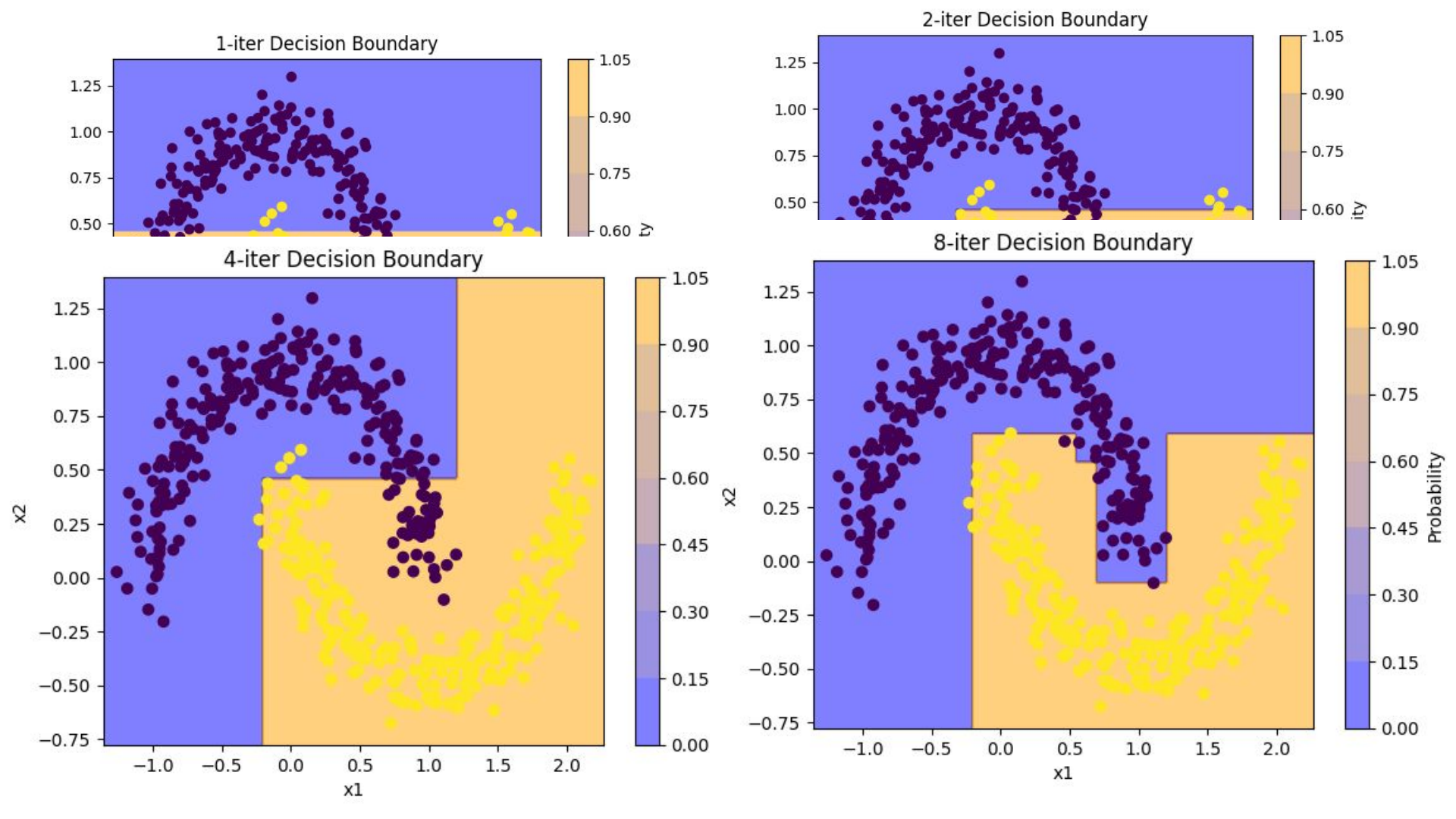
```
Model Weights: 1.147
```

```
Model Weights: -0.215
```




1-iter Decision Boundary
x2
x1
Probability
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
1.05
0.90
0.75
0.60
0.45
0.30
0.15
0.00


2-iter Decision Boundary
x2
x1
Probability
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
1.05
0.90
0.75
0.60
0.45
0.30
0.15
0.00


1-iter Decision Boundary
2-iter Decision Boundary
4-iter Decision Boundary
8-iter Decision Boundary
x1
x2
Probability

Decomposing Exp Loss

$$L(y, f(x)) = \exp\{-yf(x)\}$$

$$= \exp\left\{-y\left(\sum_{t=1}^{n} a_t h_t(x)\right)\right\}$$

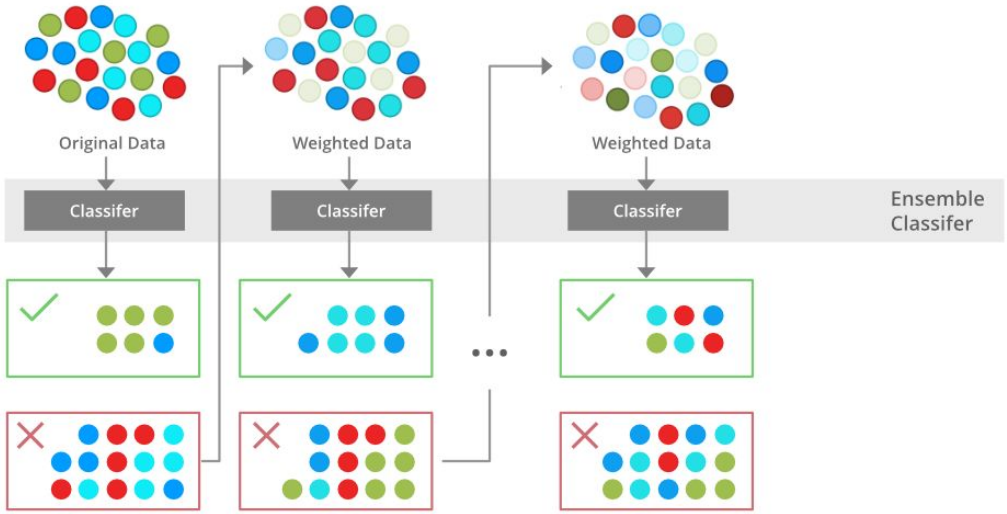
$$= \prod_{t=1}^{n} \exp\{-y a_t h_t(x)\}$$

$$D_{t+1}(i) = \frac{D_t(i)\exp\{-a_t y_i h_t(x_i)\}}{Z_t}$$

Distribution Update Rule!

```
# Update instance weights
instance_weights *= np.exp(predictor_weight * (incorrect * 2 - 1))
instance_weights = instance_weights / np.sum(instance_weights)
```

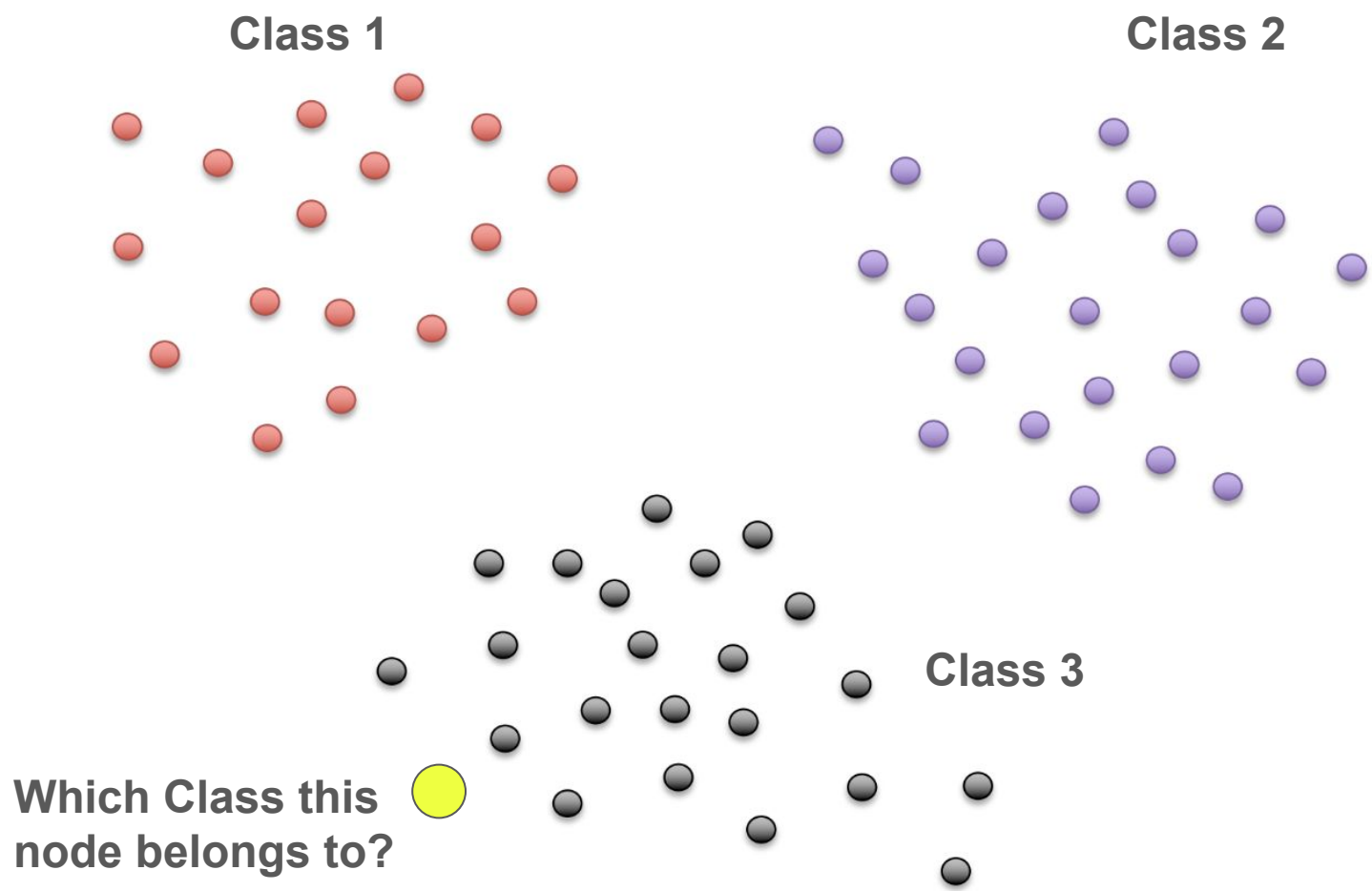
Summary of Adaboost



- Similar to last layer of MLP, Ensemble-based method also combines multiple classifier together
- Key intuition of Adaboost is that it based on the ensemble learner so far to **upvote those misclassified** data-points, and create a new (weighted) dataset to train next-iteration model.
- Though adaboost itself is not very practical now, but this **intuition** is very useful!

K-Nearest Neighbor

Simplest Supervised “Learning”


Class 1
Class 2
Class 3
Which Class this node belongs to?

Nearest Neighbors

- Suppose we're given a novel input vector $\mathbf{x}$ we'd like to classify.
- The idea: find the nearest input vector to $\mathbf{x}$ in the training set and copy its label.
- Can formalize "nearest" in terms of Euclidean distance

$$||\mathbf{x}^{(a)} - \mathbf{x}^{(b)}||_2 = \sqrt{\sum_{j=1}^{d} (x_j^{(a)} - x_j^{(b)})^2}$$

Algorithm:

1. Find example $(\mathbf{x}^*, t^*)$ (from the stored training set) closest to $\mathbf{x}$. That is:

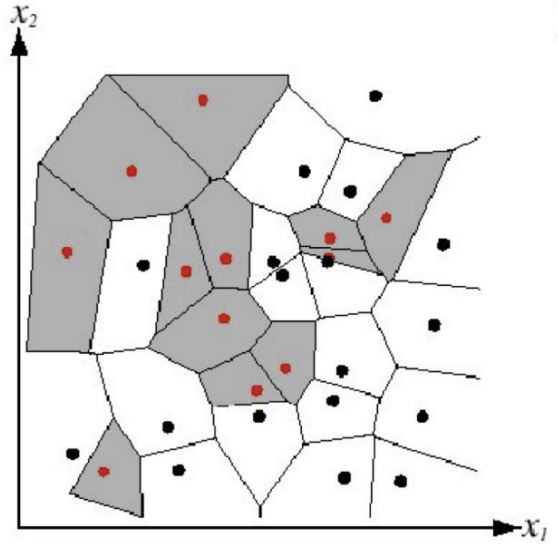
$$\mathbf{x}^* = \underset{\mathbf{x}^{(i)} \in \text{train. set}}{\text{argmin}} \text{ distance}(\mathbf{x}^{(i)}, \mathbf{x})$$

2. Output $y = t^*$

Nearest Neighbors: Decision Boundary

We can visualize the behavior in the classification setting using a Voronoi diagram.

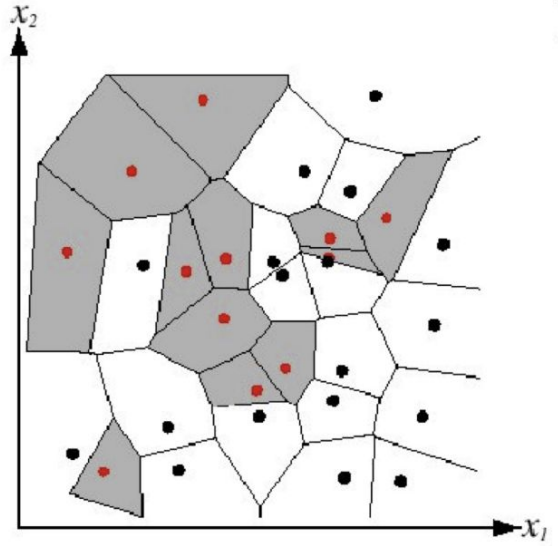
$$R_k = \{x \in X \mid d(x, P_k) \leq d(x, P_j) \text{ for all } j \neq k\}$$

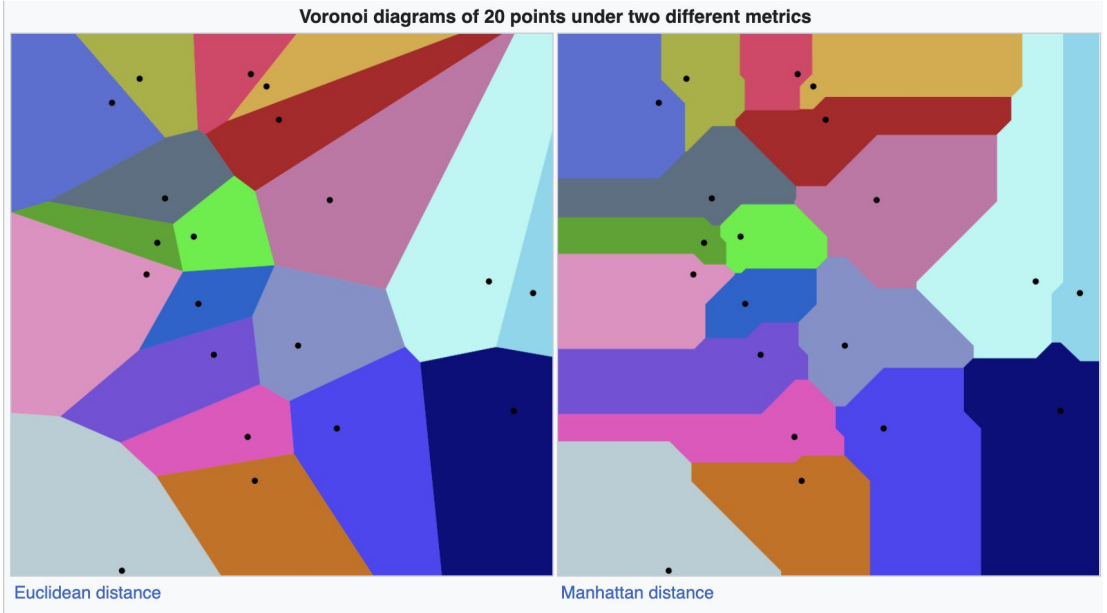


Nearest Neighbors: Decision Boundary

We can visualize the behavior in the classification setting using a Voronoi diagram.

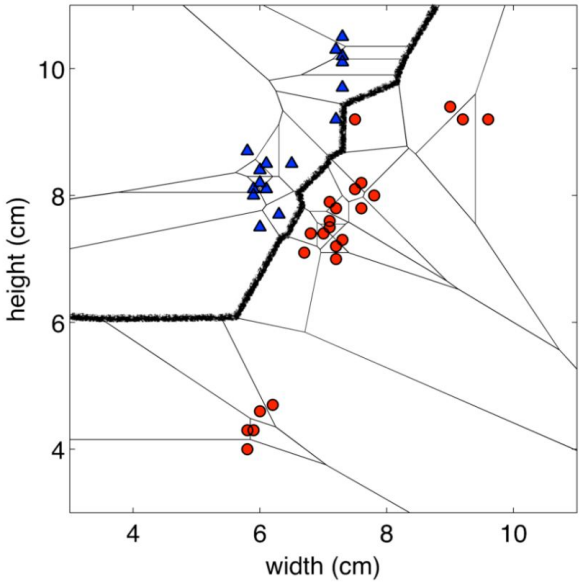
$$R_k = \{x \in X \mid d(x, P_k) \leq d(x, P_j) \text{ for all } j \neq k\}$$



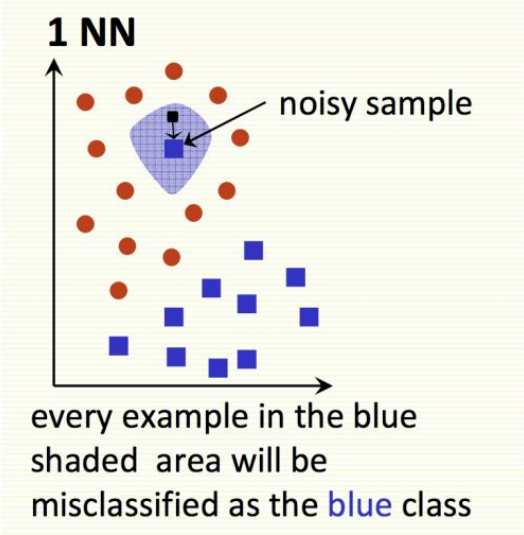


Nearest Neighbors: Decision Boundary

Decision boundary: the boundary between regions of input space assigned to different categories.



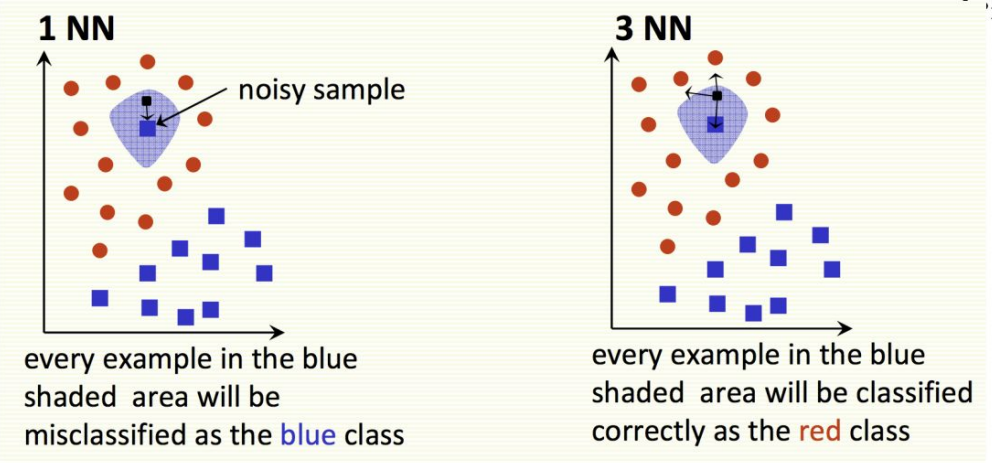
Nearest Neighbors to K-NN



[Pic by Olga Veksler]

- Nearest neighbors sensitive to noise or mis-labeled data ("class noise"). Solution?

'ic by Olga Veksler]



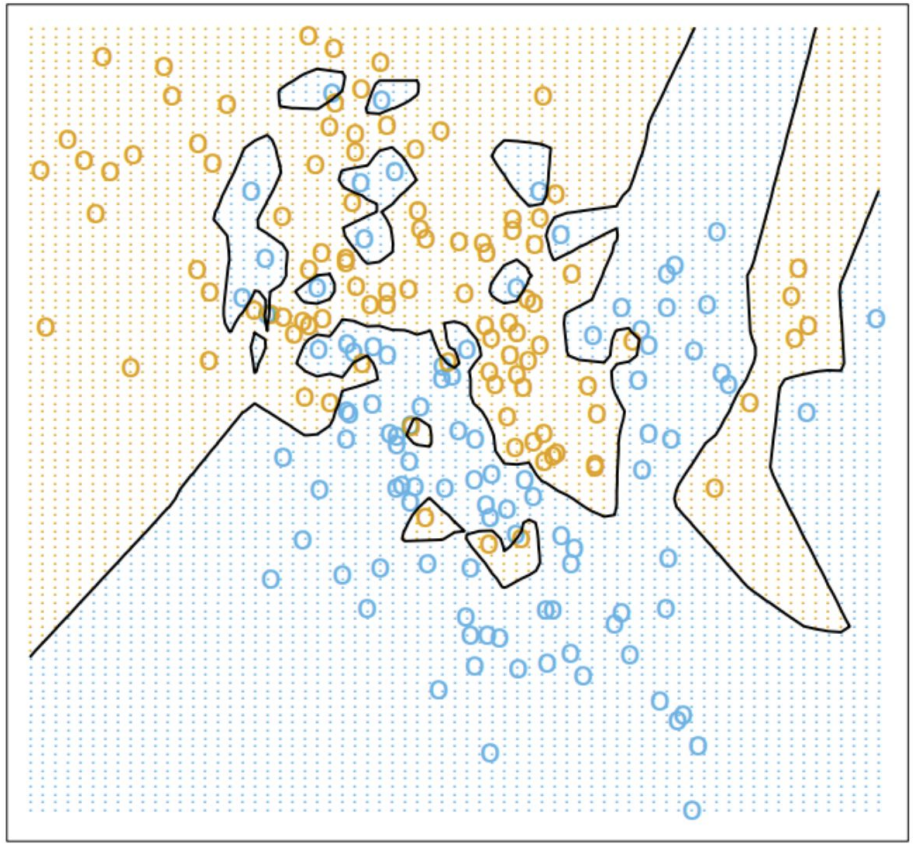
- Nearest neighbors sensitive to noise or mis-labeled data ("class noise"). Solution?
- Smooth by having k nearest neighbors vote

Algorithm (kNN):

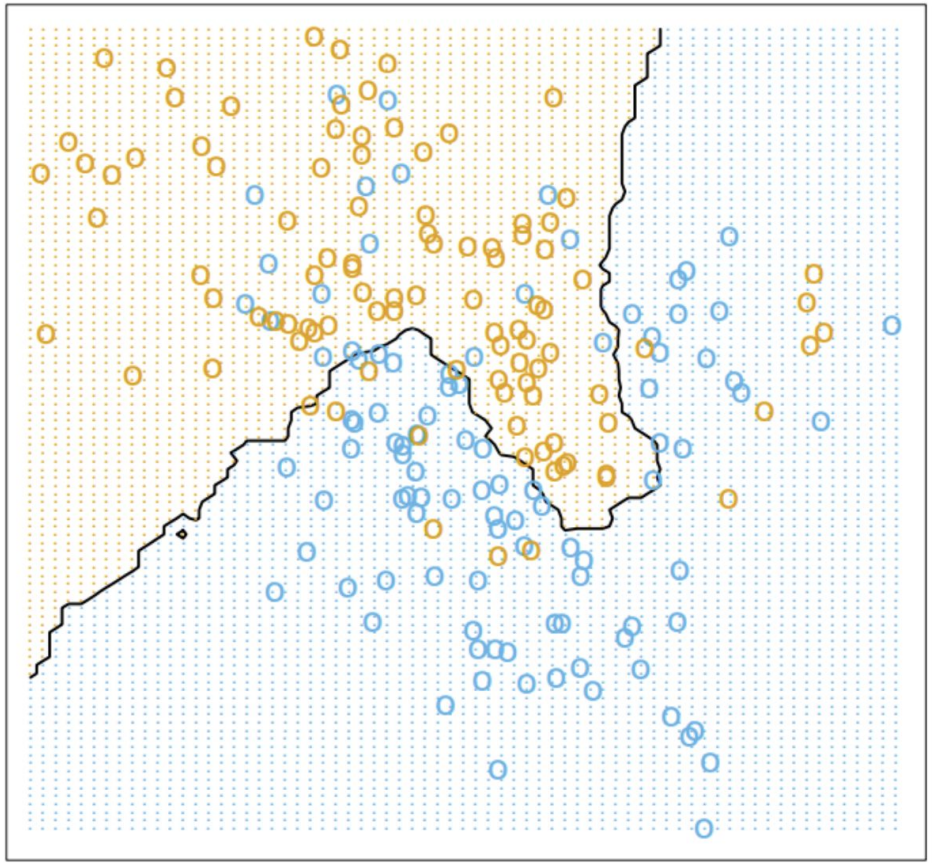
1. Find k examples $\{\mathbf{x}^{(i)}, t^{(i)}\}$ closest to the test instance $\mathbf{x}$
2. Classification output is majority class

$$y = arg\max_{t^{(z)}} \sum_{i=1}^{k} \mathbb{I}(t^{(z)} = t^{(i)})$$

K=1



K=15



K-Nearest Neighbor

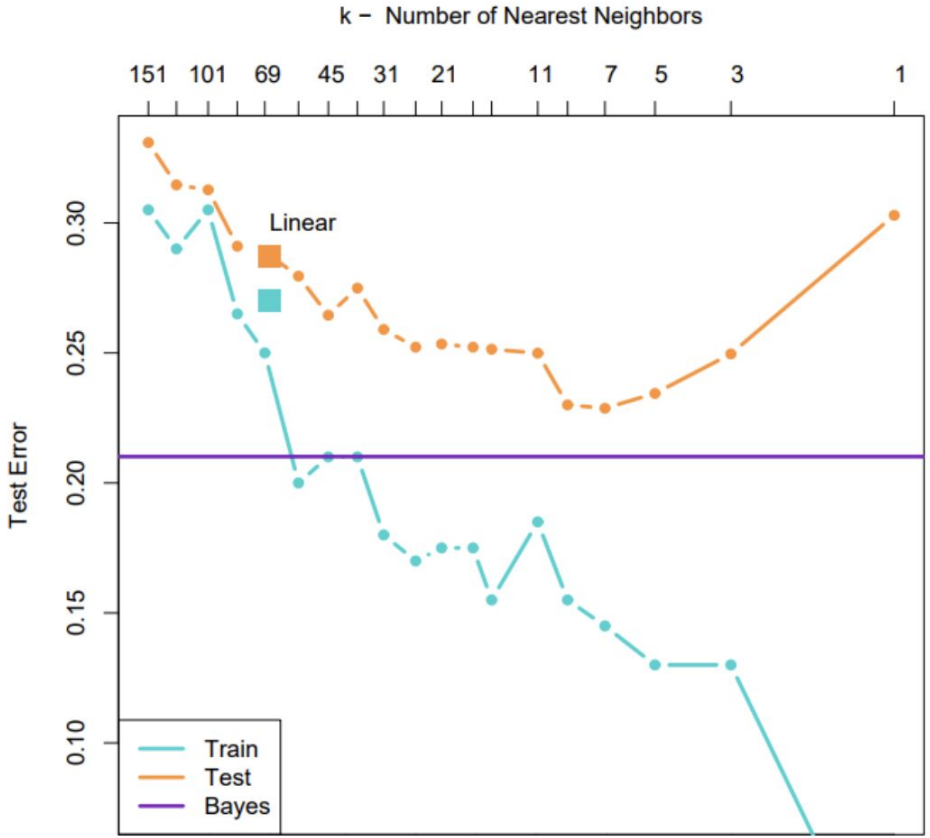
Tradeoffs in choosing k?

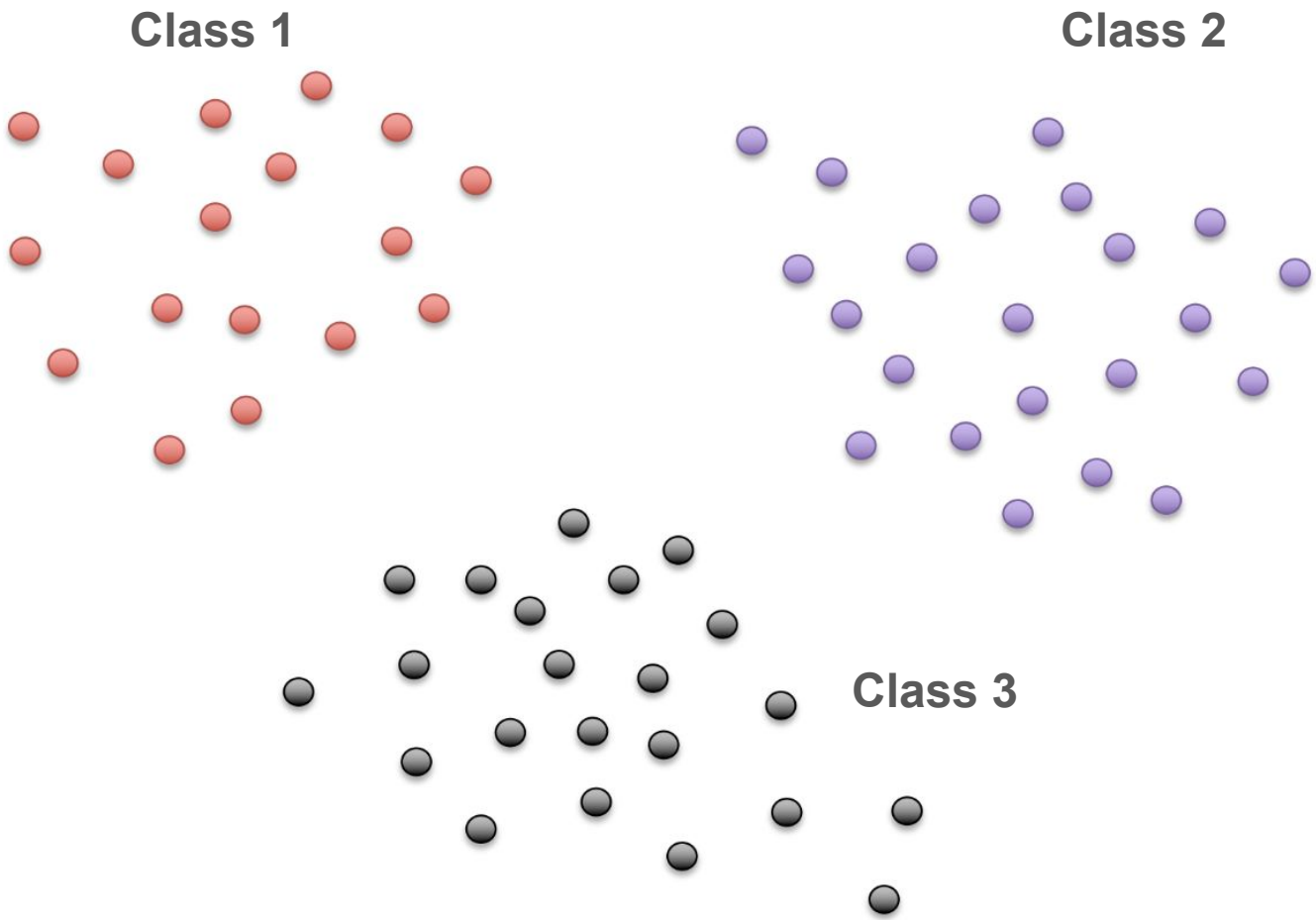
- Small k
  - Good at capturing fine-grained patterns
  - May overfit, i.e. be sensitive to random idiosyncrasies in the training data
- Large k
  - Makes stable predictions by averaging over lots of examples
  - May underfit, i.e. fail to capture important regularities
- Balancing k
  - Optimal choice of k depends on number of data points n.
  - Nice theoretical properties if $k \to \infty$ and $\frac{k}{n} \to 0$.
  - Rule of thumb: choose $k < \sqrt{n}$.

K-Nearest Neighb

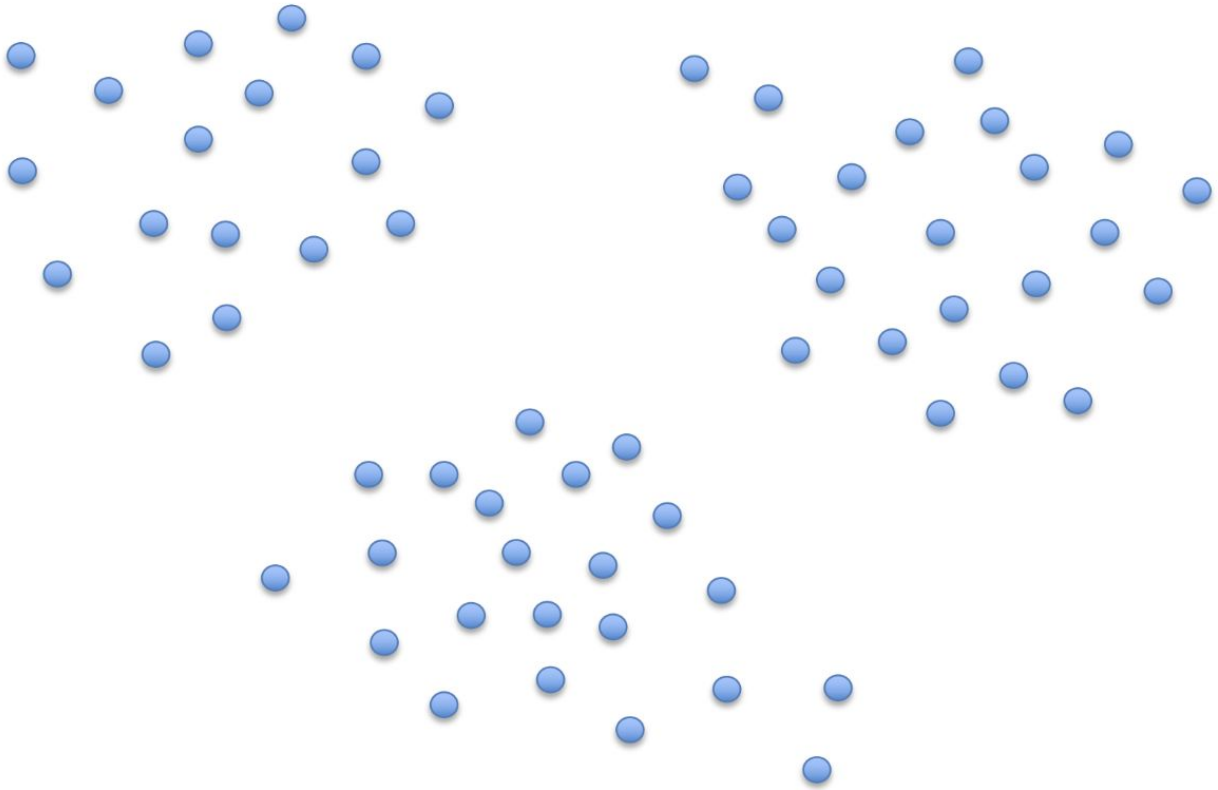
Tradeoffs in choosing k?

- Small k
  - Good at capturing fine-gr
  - May overfit, i.e. be sensit training data
- Large k
  - Makes stable predictions
  - May underfit, i.e. fail to c
- Balancing k
  - Optimal choice of k depends on number of data points n.
  - Nice theoretical properties if $k \to \infty$ and $\frac{k}{n} \to 0$.
  - Rule of thumb: choose $k < \sqrt{n}$.



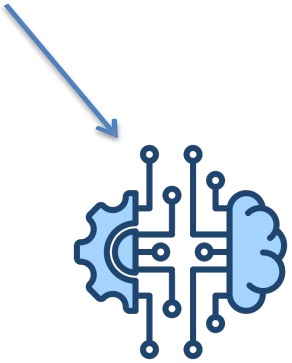

Class 1
Class 2
Class 3

What if we only observe data (X) without label (Y)?



Unsupervised Learning

Data: X



No supervised target!

Loss Functio $L(y_i, f(x_i \mid w, b))$

Learning objective is usually to reconstruction / compression of data (e.g., model P(x)).

Unsupervised Learning

- **Given:** unlabeled data:
  - Only input features
  - No labels

$$S = \{x_i\}_{i=1}^{N}$$

- **Goal:** find hidden structure/patterns
  - E.g., hidden structure is a clustering of data
  - A generative model of data P(x)
    - Discussed further in future lectures
  - I.e., a low dimensional summary of the data

Cluster Analysis

- Cluster: A collection of data objects
  - similar (or related) to one another within the same group
  - dissimilar (or unrelated) to the objects in other groups
- Cluster analysis (or *clustering, data segmentation, …*)
  - Finding similarities between data according to the characteristics found in the data and grouping similar data objects into clusters

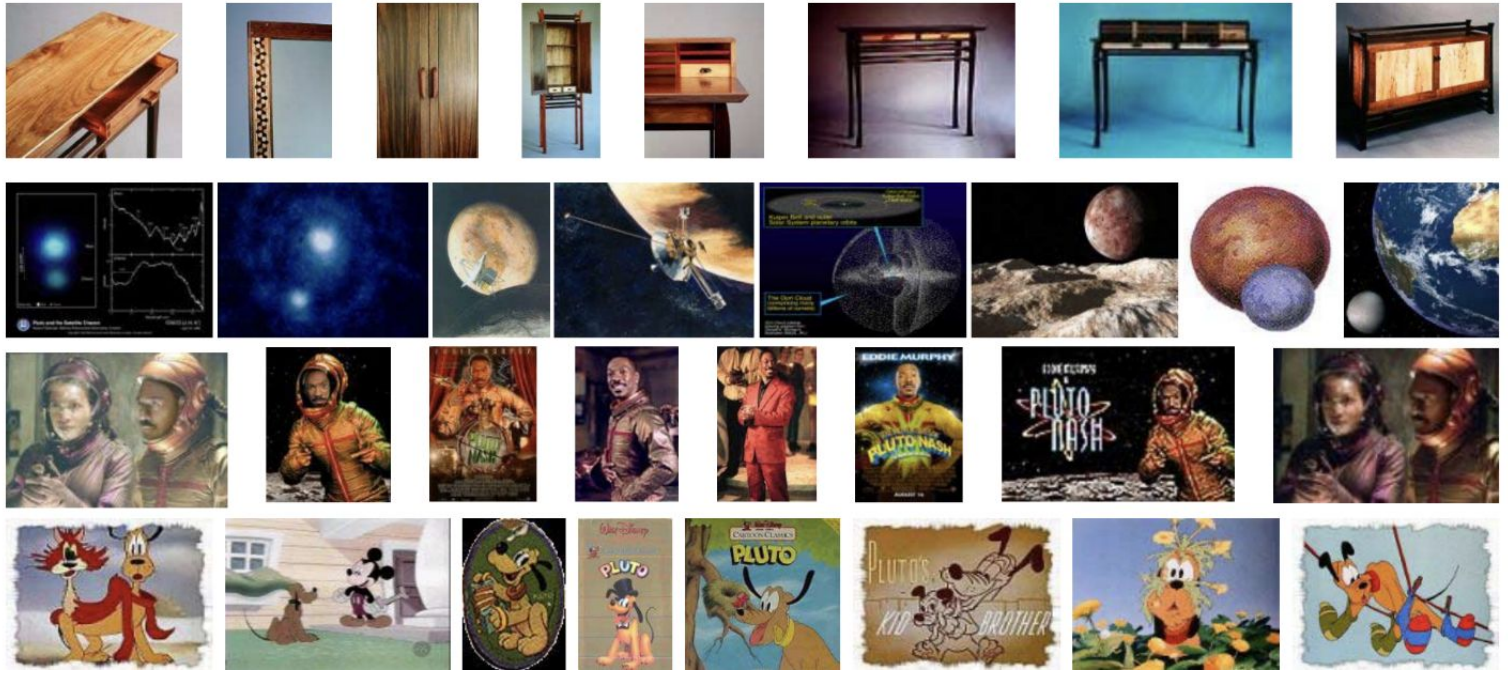
Learn / Find Clusters via **Unsupervised Learning**

Why is Clustering Useful?

- Clustering is a “summary” of data

Images about
“**Pluto**”

Each Row is a Cluster



Simplest Cluster: K-Means

- Partitioning method: Partitioning a dataset ***D*** of ***n*** objects into a set of ***k*** clusters, such that the sum of squared distances is minimized (where c_j is the centroid or medoid of cluster C_j)

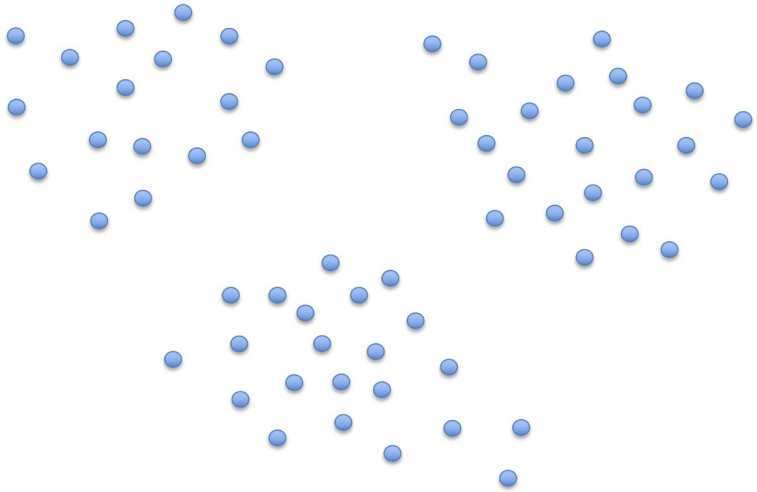
$$J = \sum_{j=1}^{k} \sum_{C(i)=j} d(x_i, c_j)^2$$

- Given *k*, find a partition of *k clusters* that optimizes the chosen partitioning criterion
  - Global optimal: exhaustively enumerate all partitions
  - Heuristic methods: *k-means* and *k-medoids* algorithms
  - *k-means* (MacQueen'67, Lloyd'57/'82): Each cluster is represented by the center of the cluster
  - *k-medoids* or PAM (Partition around medoids) (Kaufman & Rousseeuw'87): Each cluster is represented by one of the objects in the cluster

Simplest Cluster: K-Means

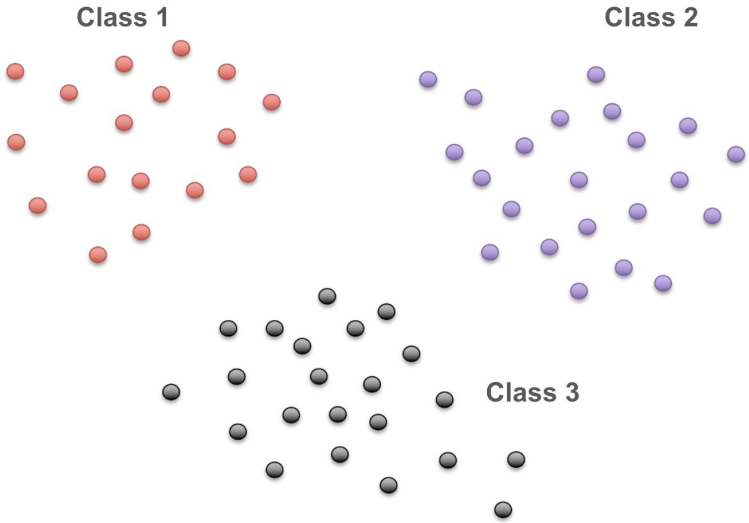
- Given *k*, the *k-means* algorithm is implemented in four steps:
  - Step 0: Partition objects into *k* nonempty subsets
  - Step 1: Compute seed points as the centroids of the clusters of the current partitioning (the centroid is the center, i.e., *mean point*, of the cluster)
  - Step 2: Assign each object to the cluster with the nearest seed point
  - Step 3: Go back to Step 1, stop when the assignment does not change

Input Data
(without label)



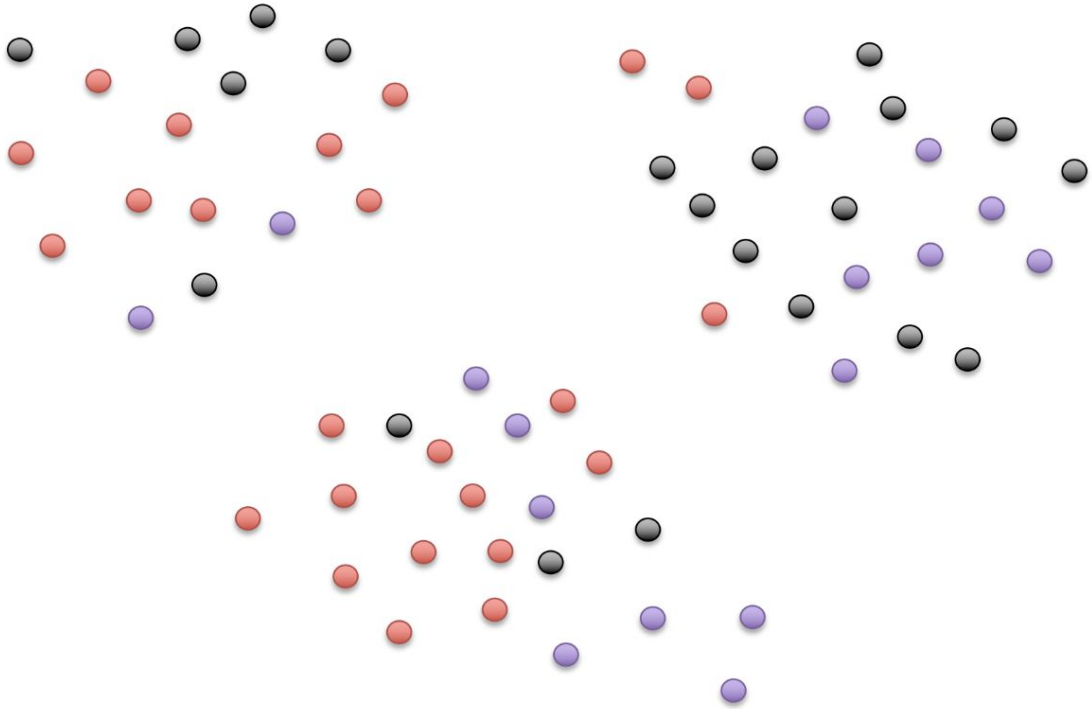
K-Means

Expected Output
(Assign Cluster Label
to each data point)



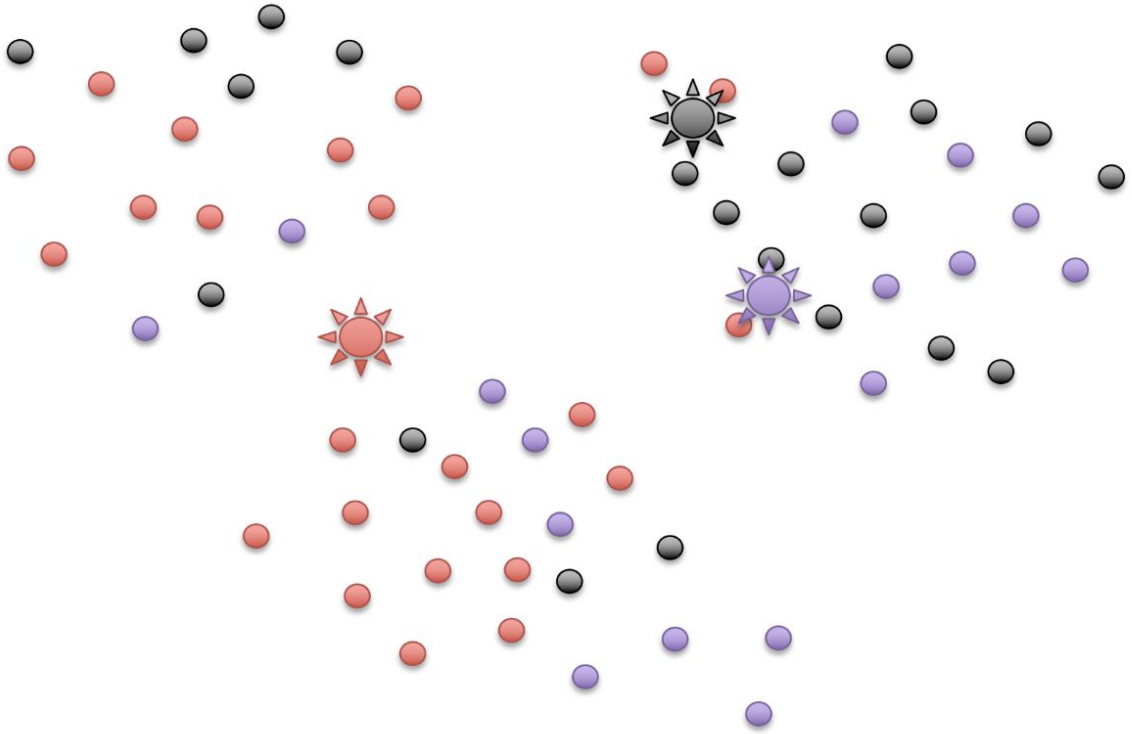
Centroid Based Clustering

(K-Means)



Initially, randomly assign label

Update Cluster Cj

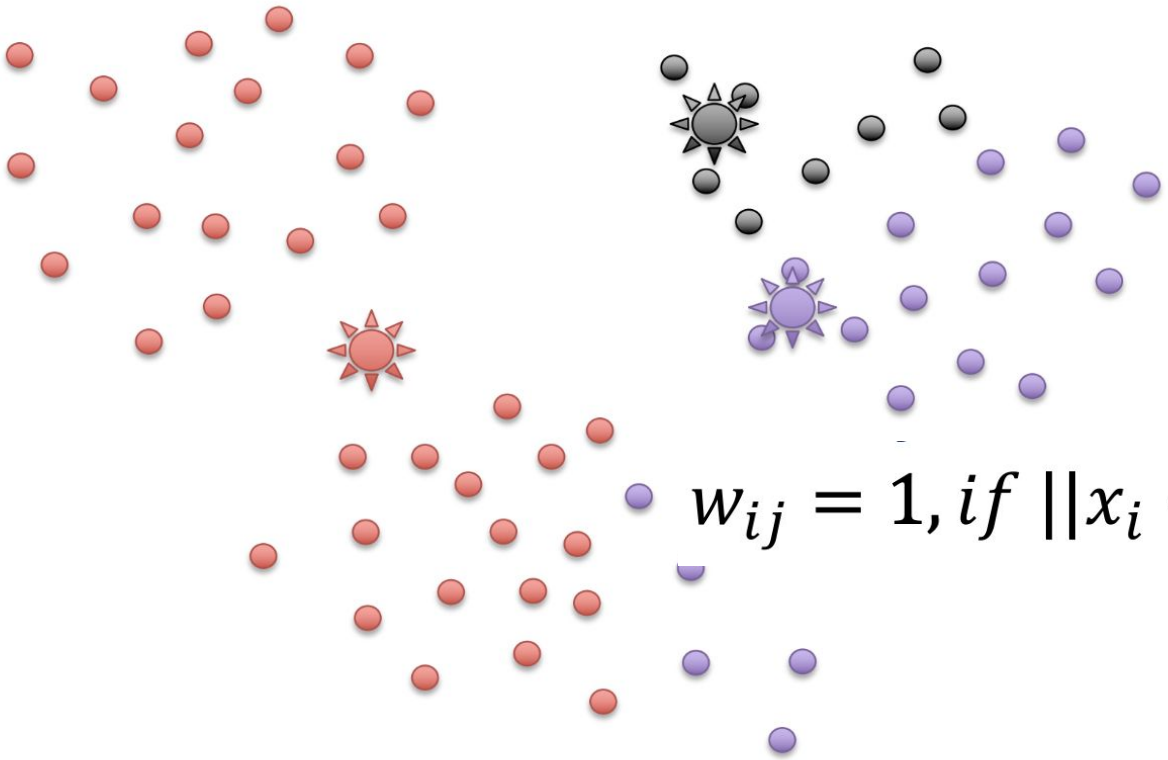


Get means (centroid, **Cj**) for each cluster j

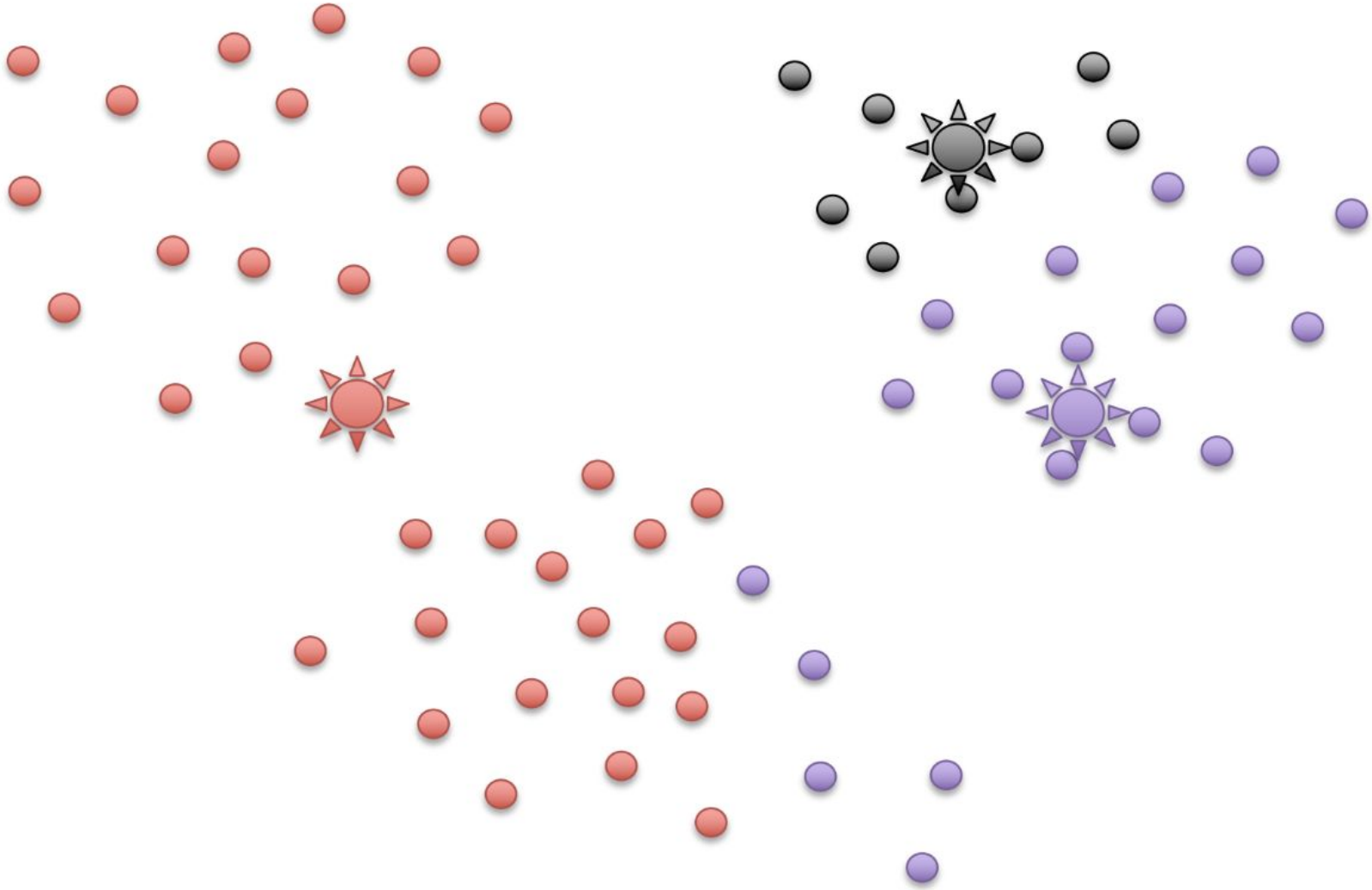
$$c_j = \frac{\sum_i w_{ij} x_i}{\sum_i w_{ij}}$$

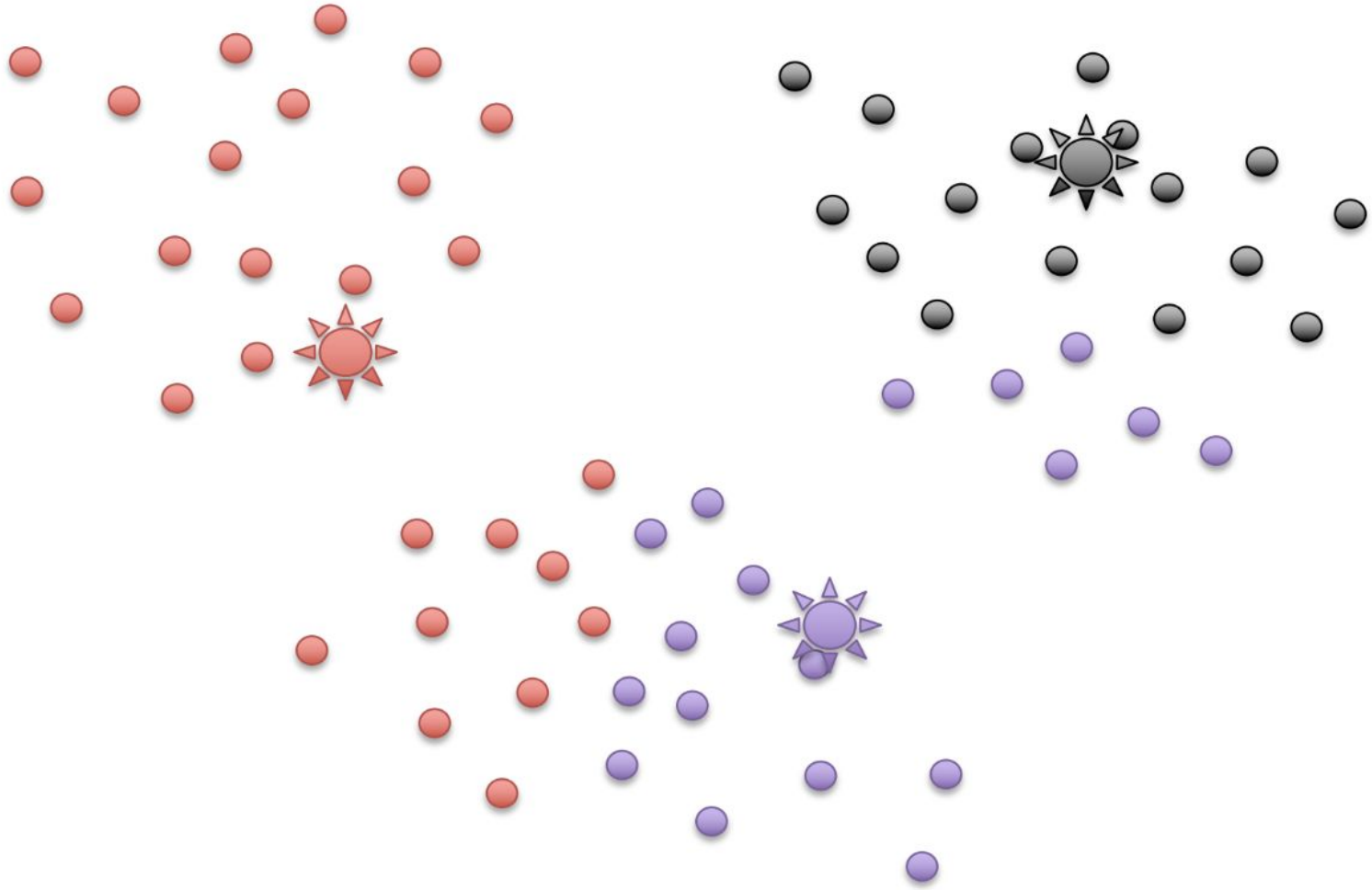
Wij = 1 if data xi belongs to cluster j

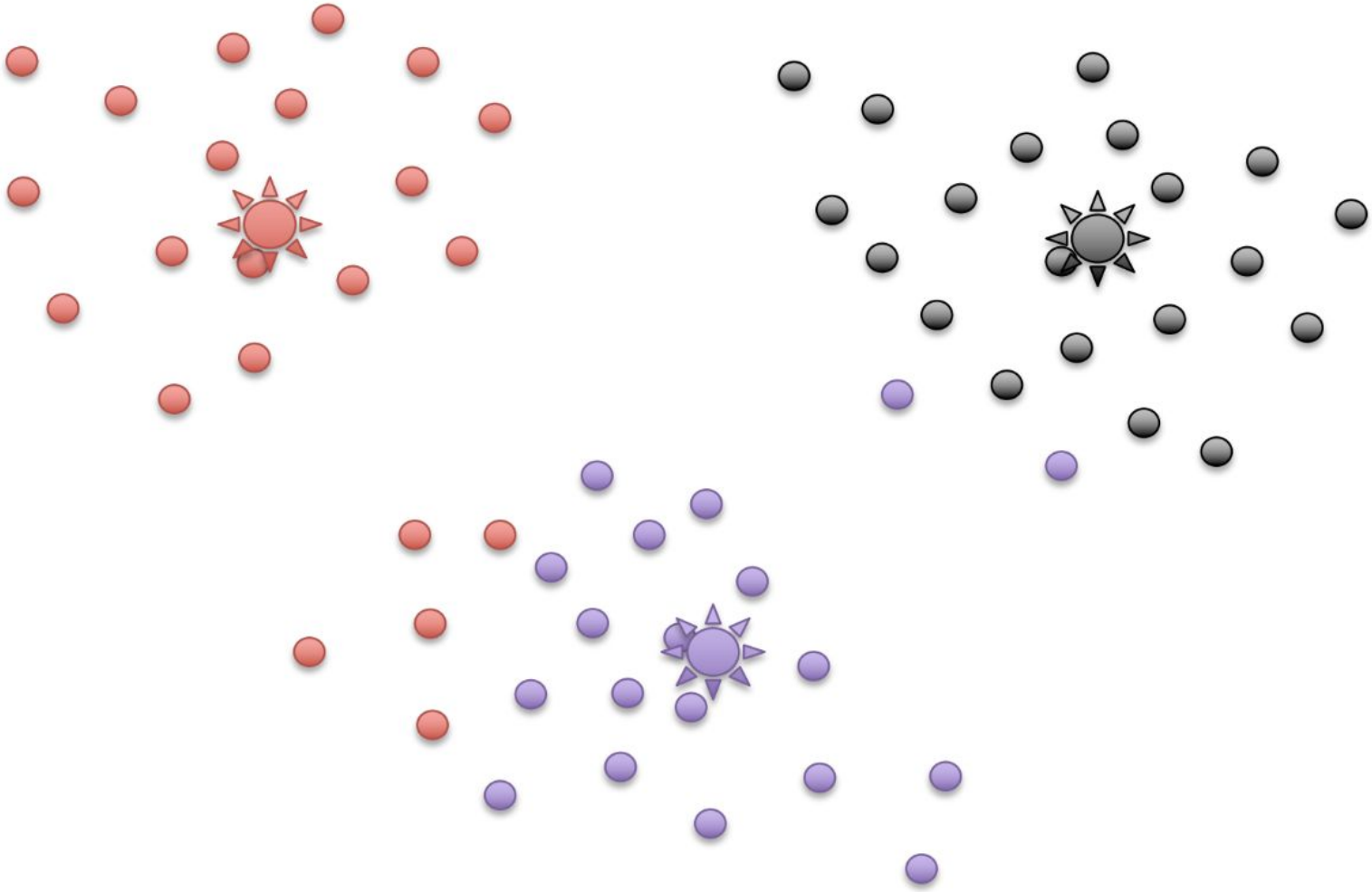
Update Wij (via K-Nearst Neighbor)

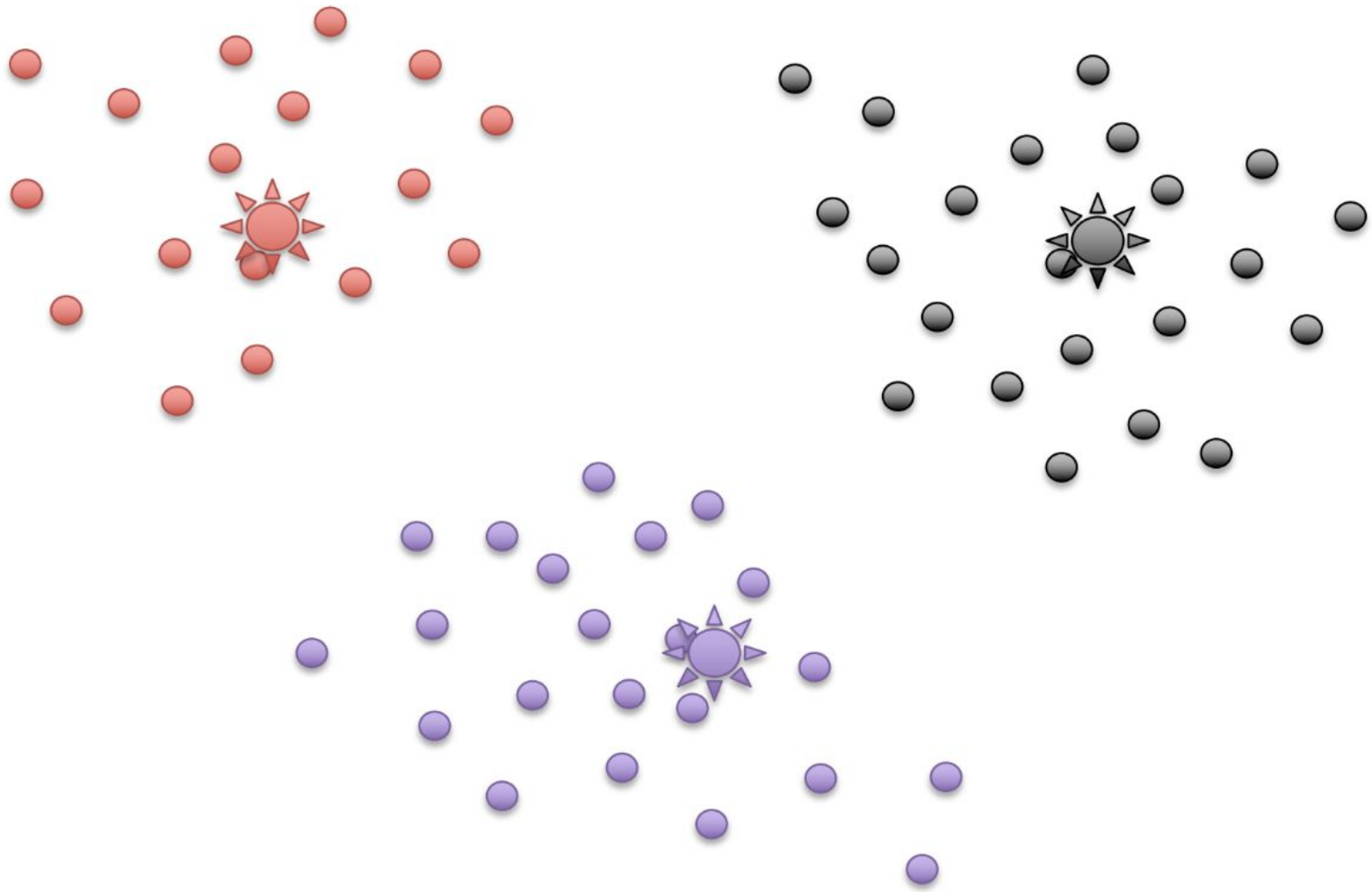


$$w_{ij} = 1, if\ ||x_i - c_j||^2 \text{ is the smallest}$$





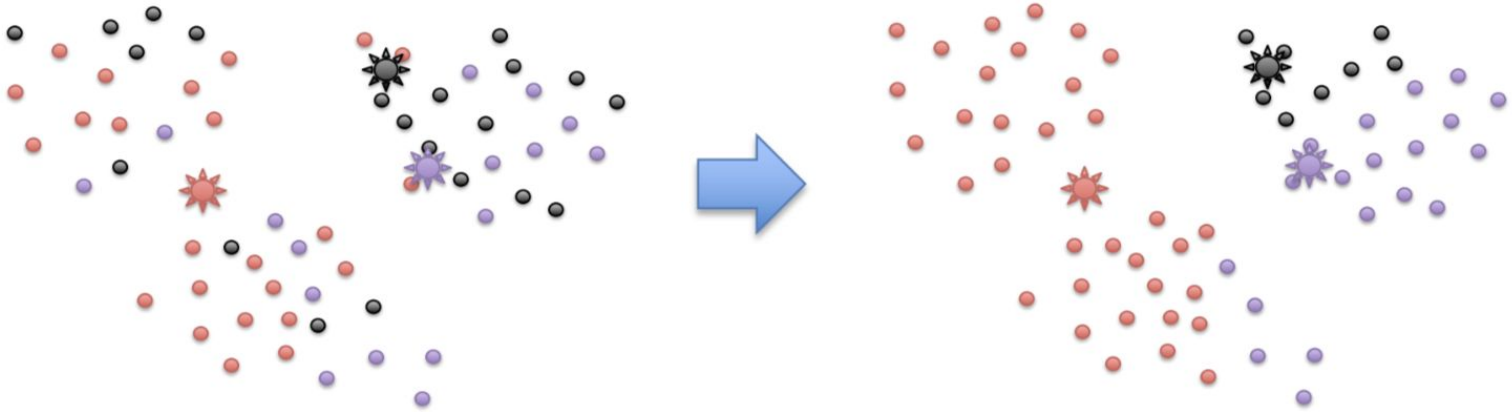




- Objective function
  - $J = \sum_{j=1}^{k} \sum_{C(i)=j} ||x_i - c_j||^2$
- Re-arrange the objective function
  - $J = \sum_{j=1}^{k} \sum_{i} w_{ij} ||x_i - c_j||^2$
    - $w_{ij} \in \{0,1\}$
    - $w_{ij} = 1, if\ x_i\ belongs\ to\ cluster\ j; w_{ij} = 0, otherwise$
    - Looking for:
      - The best assignment w_{ij}
      - The best center c_j

Step 1: Update Assignment

- For each x: $w_{ij} = 1, if\ ||x_i - c_j||^2$ is the smallest
  - Assign to cluster C_k with smallest distance to c_k



Step 1: Update Assignment

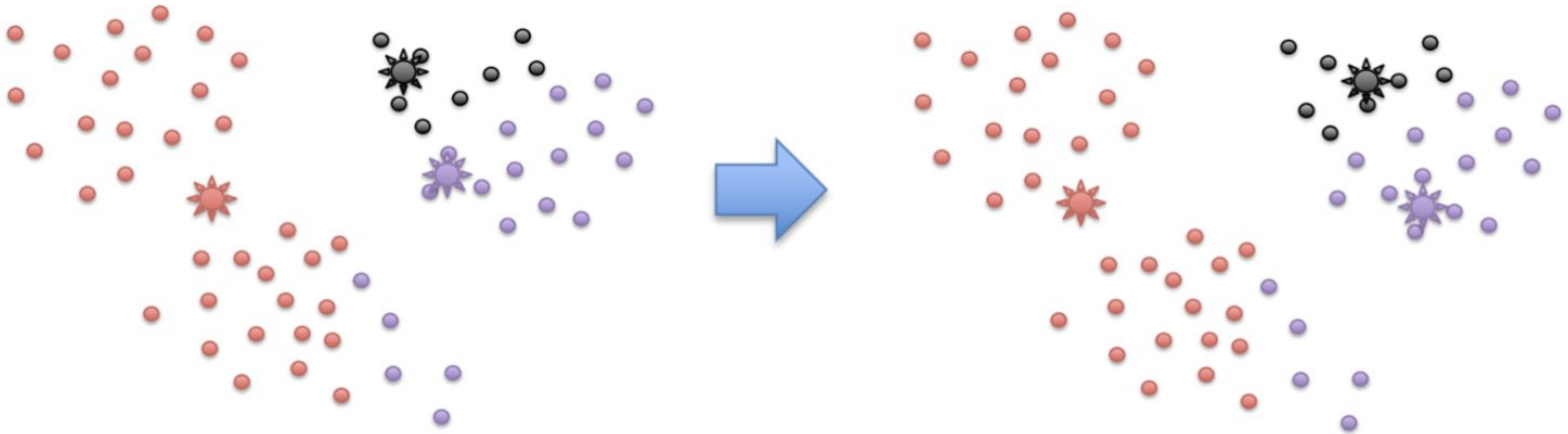
- For each x: $w_{ij} = 1, if\ ||x_i - c_j||^2$ is the smallest

- Such K-Nearest Neighbor Assignment **minimize** the cost function given cluster center (c1, c2, …) **fixed**

$$J = \sum_{j=1}^{k} \sum_{i} w_{ij} ||x_i - c_j||^2$$

Step 2: Update Cluster Center

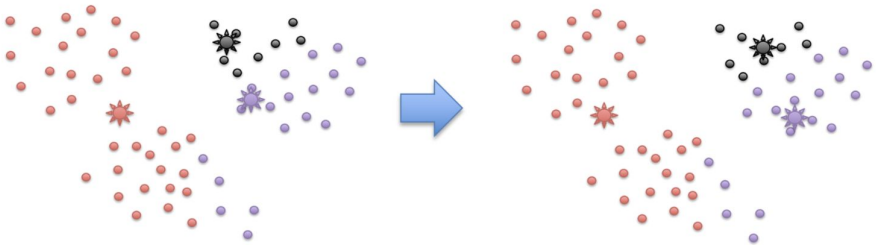
- For each c_k:
  - Compute $c_k = \text{mean}(C_k)$



Step 2: Fix assignment w_{ij}, find centers that minimize J

$$J = \sum_{j=1}^{k} \sum_{i} w_{ij} ||x_i - c_j||^2$$

- => first derivative of J = 0
- => $\frac{\partial J}{\partial c_j} = -2 \sum_i w_{ij}(x_i - c_j) = 0$
- =>$c_j = \frac{\sum_i w_{ij} x_i}{\sum_i w_{ij}}$
  - Note $\sum_i w_{ij}$ is the total number of objects in cluster j

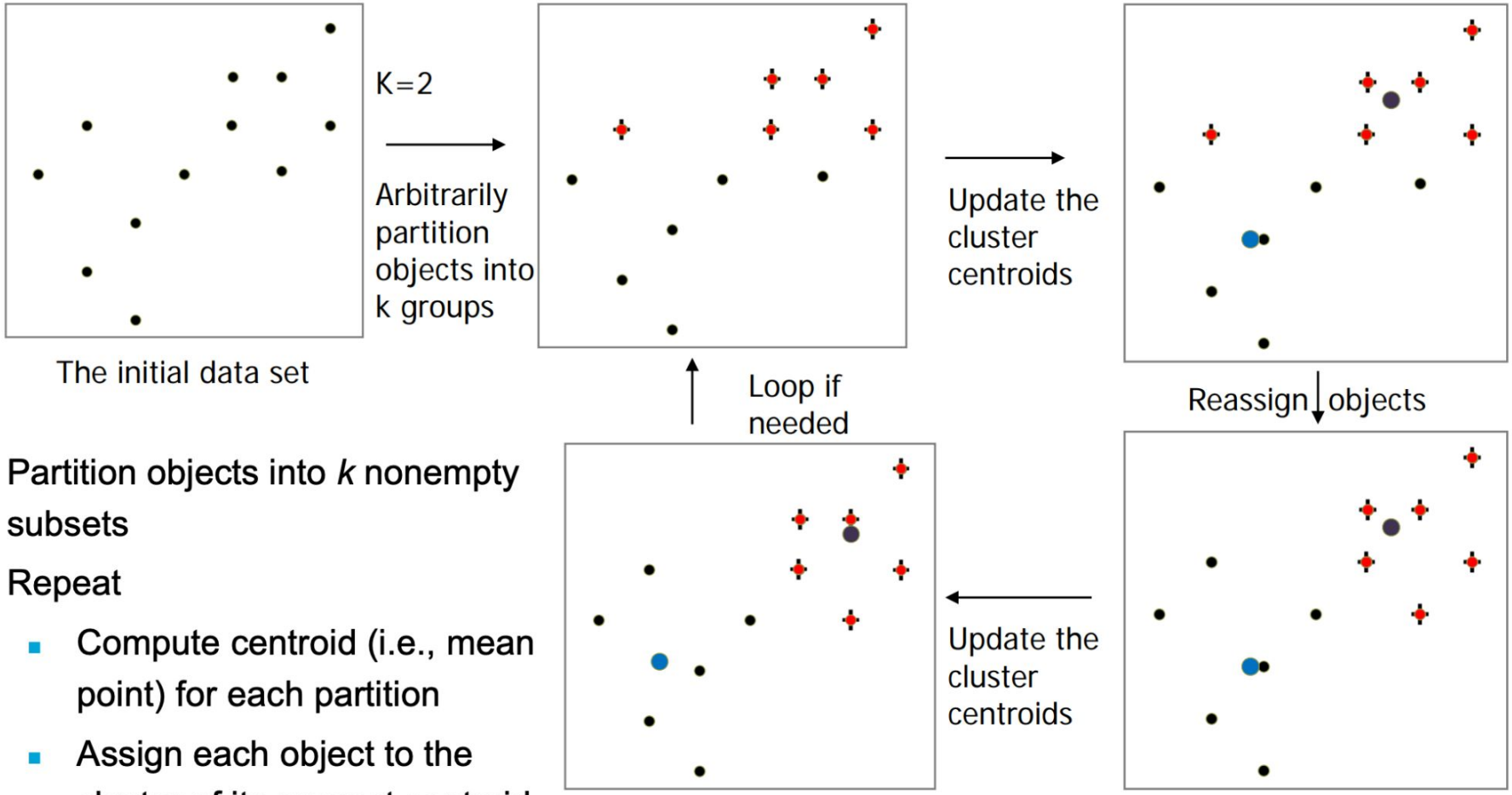


K-Means Algorithm

- Iterations

$$J = \sum_{j=1}^{k} \sum_{i} w_{ij} ||x_i - c_j||^2$$

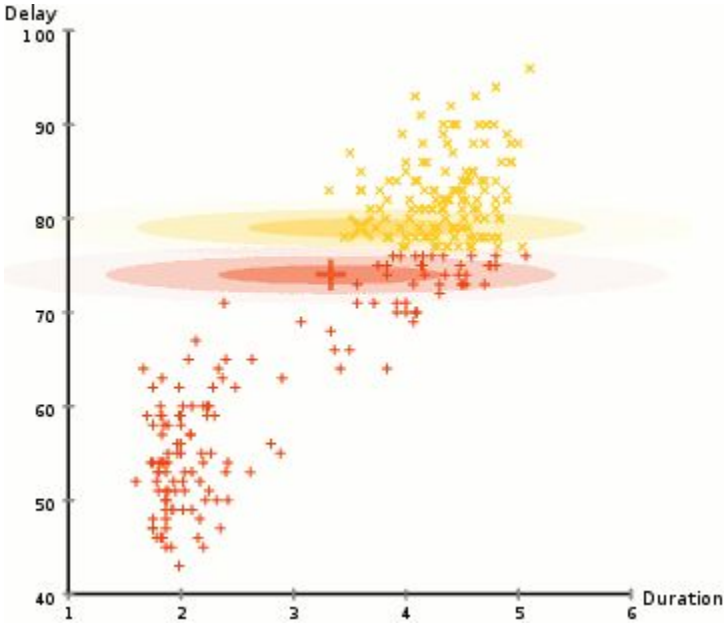
  - Step 1: Fix centers c_j, find assignment w_{ij} that minimizes J
    - => $w_{ij} = 1, if\ ||x_i - c_j||^2$ is the smallest
  - Step 2: Fix assignment w_{ij}, find centers that minimize J
    - => first derivative of J = 0
    - => $\frac{\partial J}{\partial c_j} = -2 \sum_i w_{ij}(x_i - c_j) = 0$
    - => $c_j = \frac{\sum_i w_{ij} x_i}{\sum_i w_{ij}}$
      - Note $\sum_i w_{ij}$ is the total number of objects in cluster j



- Partition objects into *k* nonempty subsets
- Repeat
  - Compute centroid (i.e., mean point) for each partition
  - Assign each object to the cluster of its nearest centroid
- Until no change

K-Means belongs to EM Algorithm

- Expectation–maximization (EM) algorithm is an iterative method to find maximum likelihood a model with unknown latent variable
- In Cluster setup, unknown variable is cluster assignment W_ij.



K-Means belongs to EM Algorithm

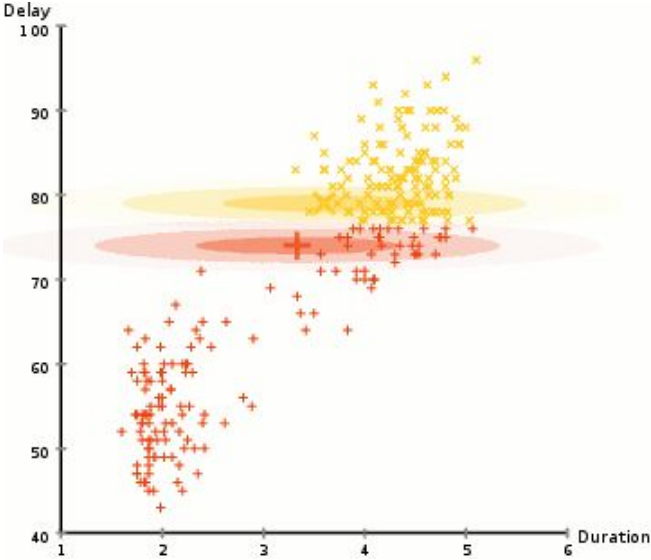
- For both steps, we aim to minimize J
- Expectation (E-Step)
  - Estimate cluster assignment W_ij given a fixed cluster center c_j

$$w_{ij} = 1, if\ ||x_i - c_j||^2 \text{ is the smallest}$$

- Maximization (M-Step)
  - Update model parameter (cluster center) by maximizing likelihood

$$\frac{\partial J}{\partial c_j} = -2\sum_i w_{ij}(x_i - c_j) = 0$$

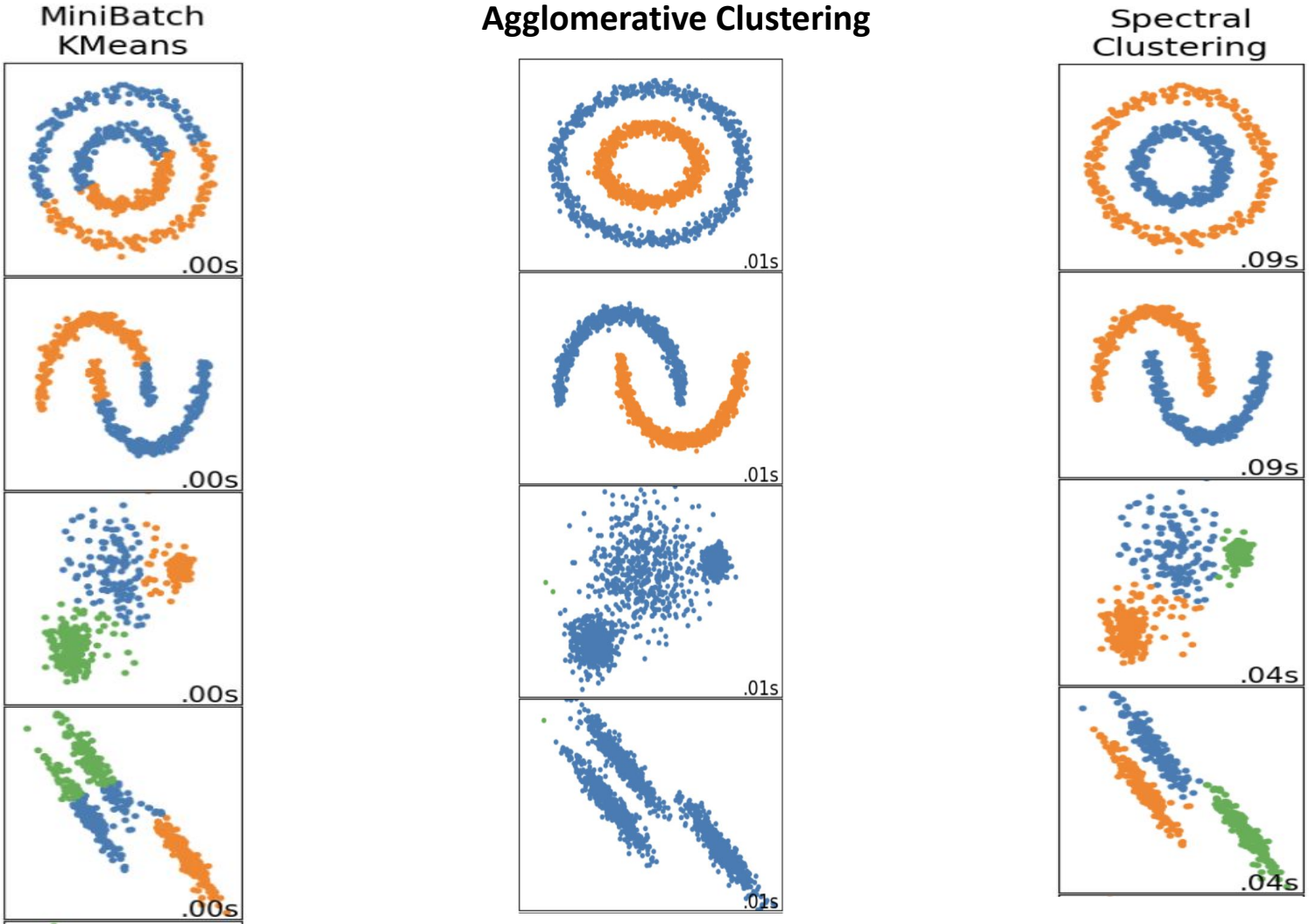
$$c_j = \frac{\sum_i w_{ij} x_i}{\sum_i w_{ij}}$$



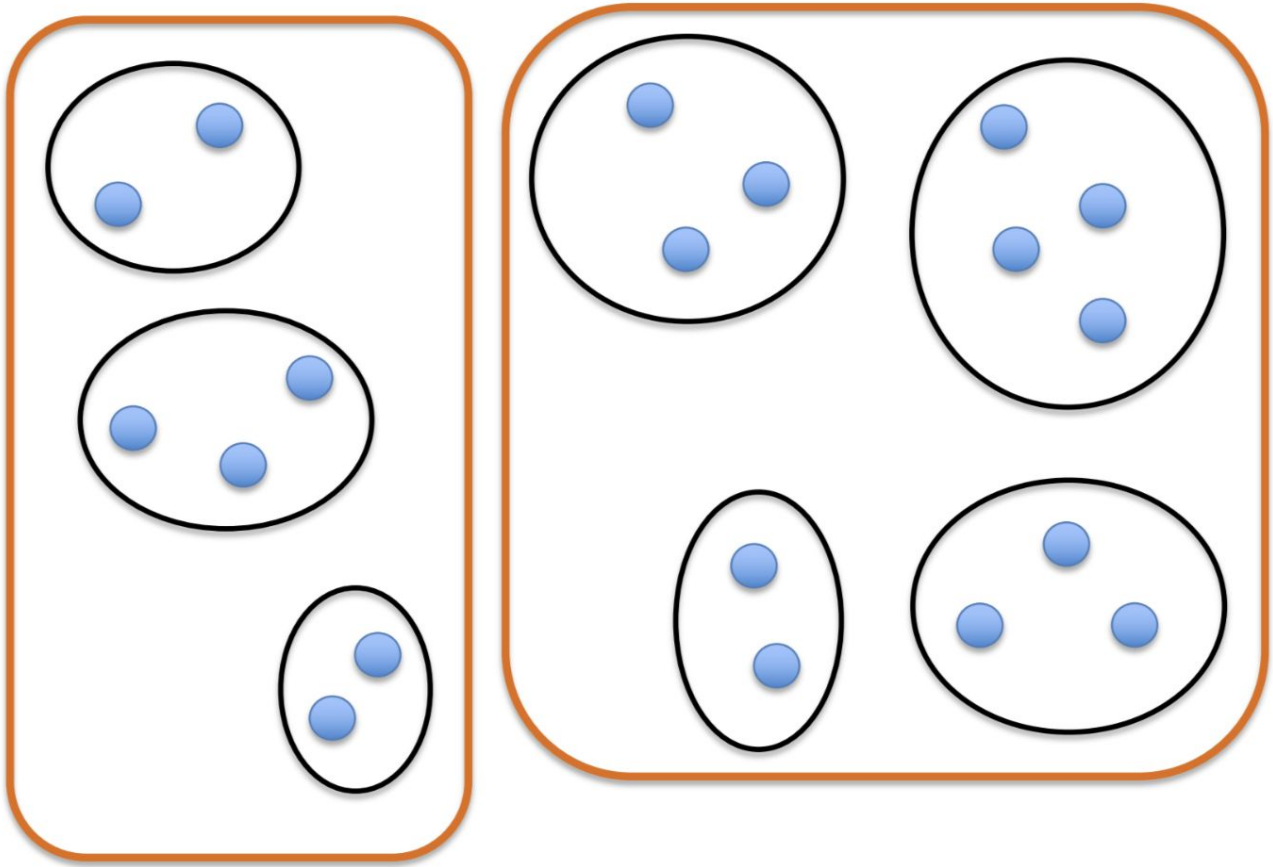
- Strength: *Efficient*: $O(tkn)$, where n is # objects, k is # clusters, and t is # iterations. Normally, $k, t << n$.
- Comment: Often terminates at a *local optimal*
- Weakness
  - Applicable only to objects in a continuous n-dimensional space
    - Using the k-modes method for categorical data
    - In comparison, k-medoids can be applied to a wide range of data

- *k-means* (MacQueen'67, Lloyd'57/'82): Each cluster is represented by the center of the cluster
- *k-medoids* or PAM (Partition around medoids) (Kaufman & Rousseeuw'87): Each cluster is represented by one of the objects in the cluster

- Strength: *Efficient*: $O(tkn)$, where n is # objects, k is # clusters, and t is # iterations. Normally, $k, t << n$.
- Comment: Often terminates at a *local optimal*
- Weakness
  - Applicable only to objects in a continuous n-dimensional space
    - Using the k-modes method for categorical data
    - In comparison, k-medoids can be applied to a wide range of data
  - Need to specify *k*, the *number* of clusters, in advance (there are ways to automatically determine the best k (see Hastie et al., 2009)
  - Sensitive to noisy data and *outliers*
  - Not suitable to discover clusters with *non-convex shapes*


MiniBatch KMeans
Agglomerative Clustering
Spectral Clustering
.00s
.00s
.00s
.00s
.01s
.01s
.01s
.01s
.09s
.09s
.04s
.04s

What is good clustering?



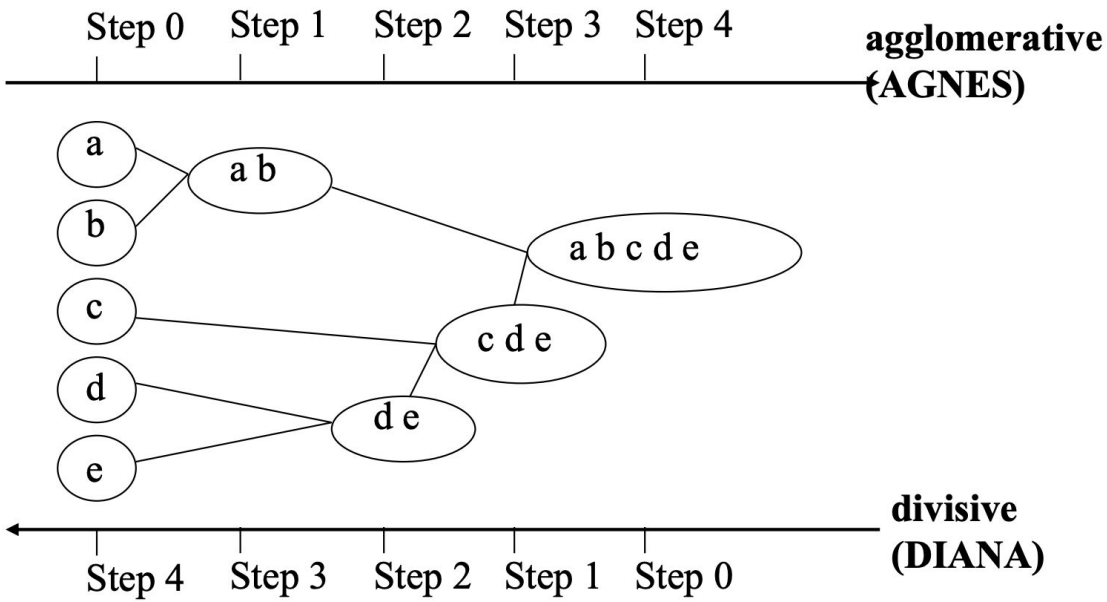
Linkage Based Clustering
(Hierarchical Clustering)

- K-Means used centroid clustering structure
  - Clustered data points are “close” to cluster center

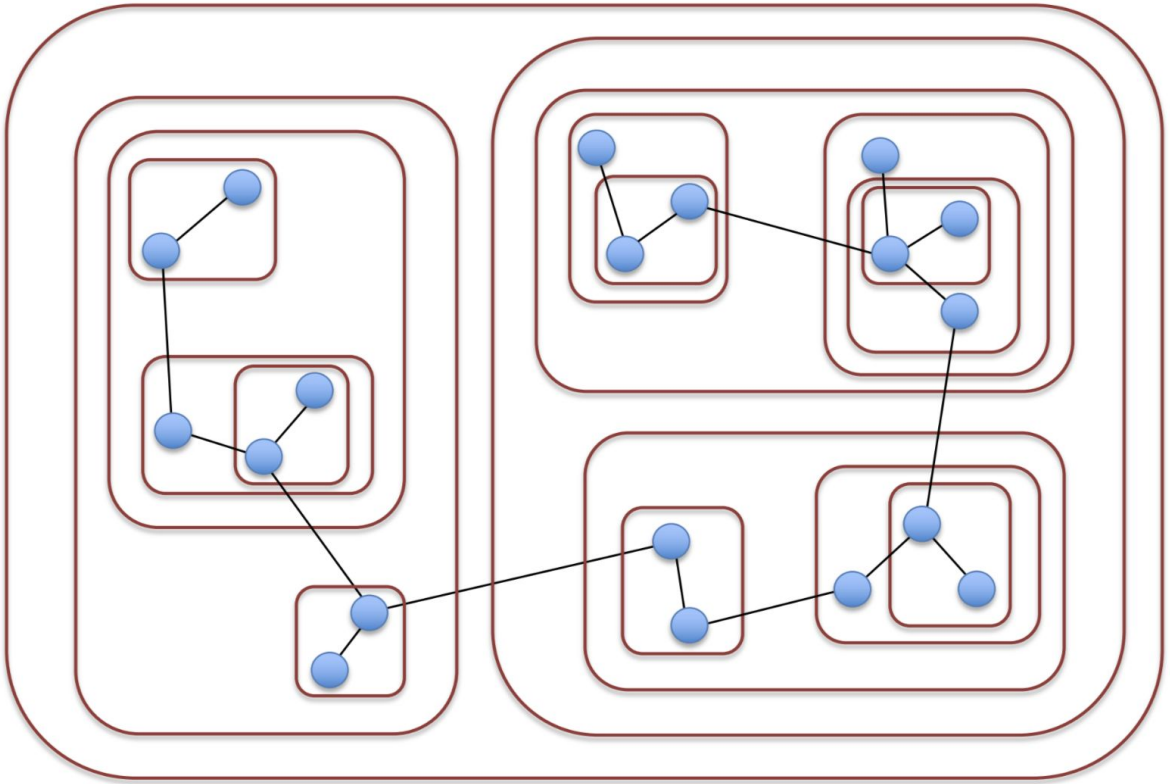
- Sometimes a linkage structure is better…
  - Employ hierarchical clustering
  - E.g., agglomerative clustering

Hierarchical Clustering

- Use distance matrix as clustering criteria. This method does not require the number of clusters ***k*** as an input, but needs a termination condition

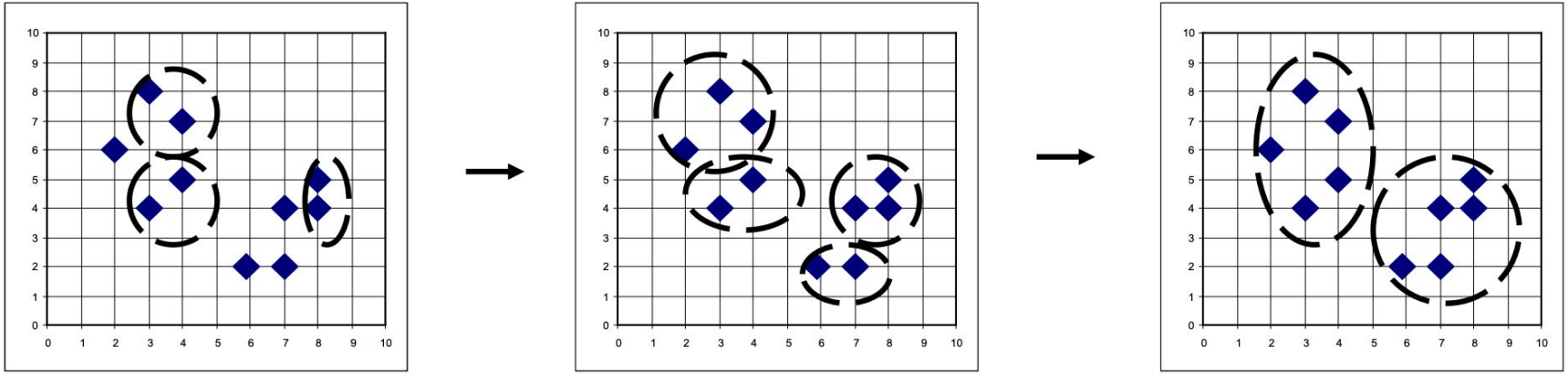


Hierarchical Clustering



Agglomerative Clustering

- Use the **single-link** method and the dissimilarity matrix
- Merge nodes that have the least dissimilarity
- Go on in a non-descending fashion
- Eventually all nodes belong to the same cluster



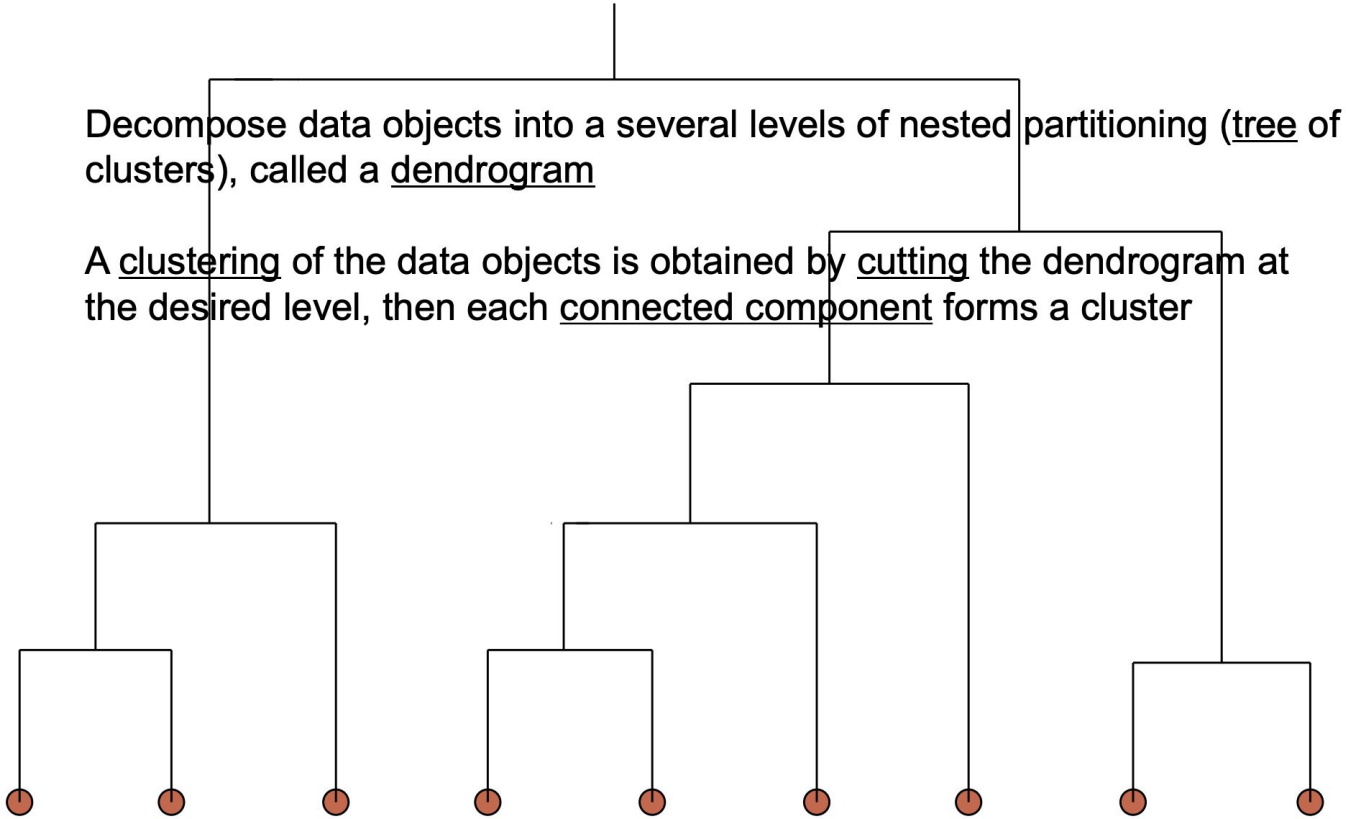
Pseudo Code

- Initialization: Place each data point into its own cluster and compute distance matrix between clusters
- Repeat:
  - Merge the two closest clusters
  - Update the distance matrix for the affected entries
- Until: all the data are merged into a single cluster

Dendrogram: Shows How Clusters are Merged

Decompose data objects into a several levels of nested partitioning (tree of clusters), called a dendrogram

A clustering of the data objects is obtained by cutting the dendrogram at the desired level, then each connected component forms a cluster



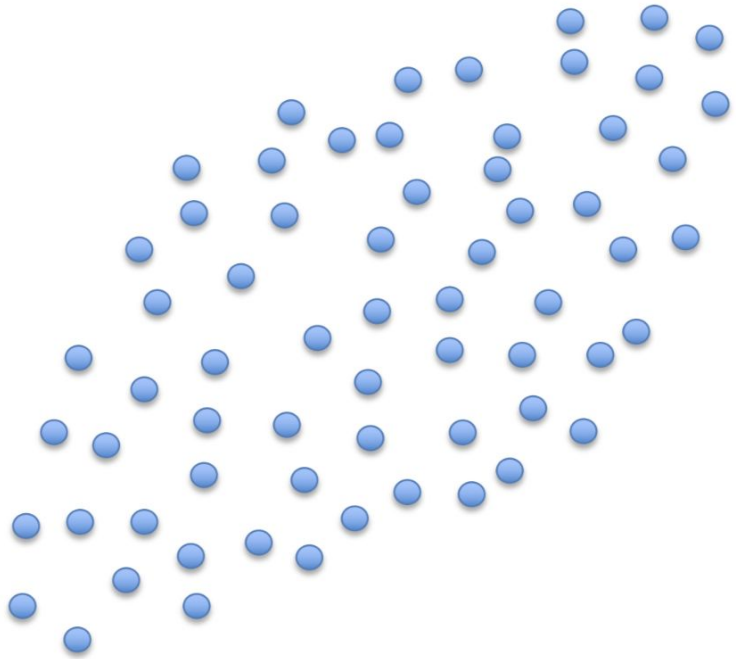
Agglomerative Clustering

- Equivalent to finding minimum spanning tree
  - Kruskal's Algorithm
  - http://en.wikipedia.org/wiki/Kruskal%27s_algorithm
- Order that edges are added defines the cluster hierarchy
- Equivalent to finding a binary tree partitioning with progressively smaller partition distances

Recap: Clustering

- Unsupervised learning
  - Finds the clustering structure of input features
- Centroid based
  - Clusters should be clumped together
  - K-Means
- Linkage Based
  - Clusters can be organized hierarchically
  - Agglomerative Clustering
- Works great when clustering assumption is good!

Limitations of Clustering



Assume to have separable feature

Need a better way to automatically learn such feature (representation)

- Neural Network
- Dimension Deduction (introduce later)