Advanced Data Analysis

UE Master 2 SCI20

N. Usunier nicolas.usunier@utc.fr

Overview of this lecture

- Lecture II:
  - First supervised learning algorithms: k-nearest neighbors and decision trees
  - Unsupervised learning algorithms: clustering methods

k-nearest neighbors algorithm

k-Nearest Neighbors Algorithm (1/2)

- Probably the simplest supervised learning algorithm
- Mostly designed for classification

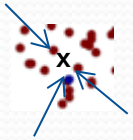
  (but versions exist for regression)
- Widely studied in statistical learning theory

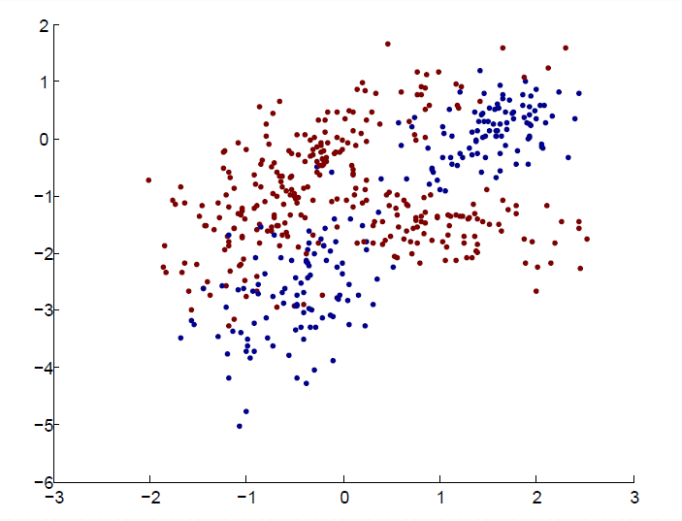
  (strong asymptotic guarantees)
- *k*-nn is a *memory-based* method:
  - No "learning" phase
  - Prediction on a new instance is performed using the labels of similar instances of the training set

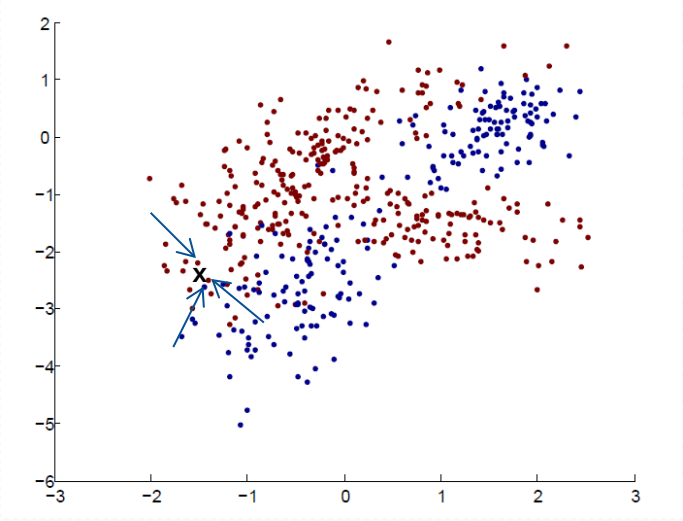
k-Nearest Neighbors Algorithm (2/2)

- Input:
  - Training dataset $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$
    - Each $x_i \in R^d$ belongs to a metric space
      (usually $x_i \in R^d$, with euclidian distance)
    - The y_i are the class labels (i.e. $y_i \in \{-1, 1\}$ for binary classification)
  - Number of neighbors: k
- Learning step: none
- Prediction algorithm:
  - Input: new instance x
  - Algorithm: find the k examples in S that are closest to x
  - Output: most frequent class labels in these k neighbors

k-NN Algorithm Example

Zoom: 





Training data
(500 examples)

red: class -1
blue: class 1

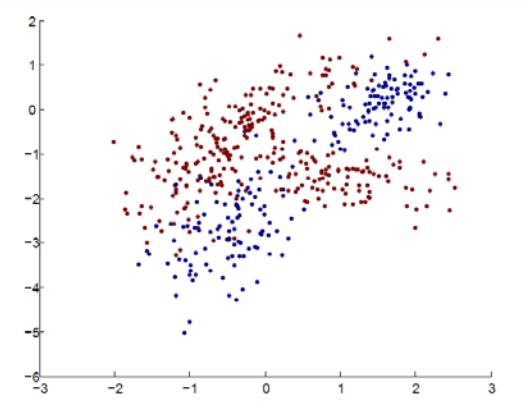
black cross: new instance
(arrows point to nearest neighbors)

Predicted class: -1

k-NN Algorithm: the Effect of *k*

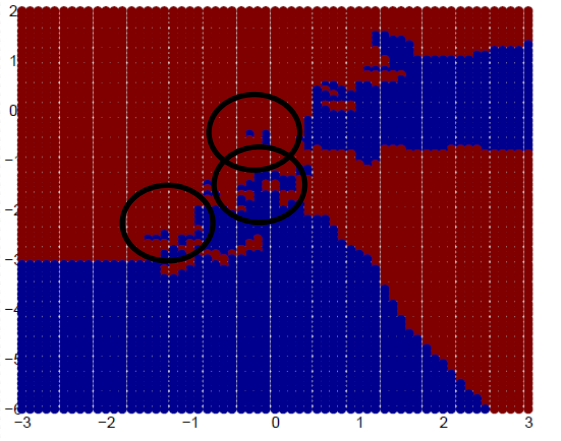
- *k* plays the role of a capacity control parameter

(in reality: *k* is a s"smoothing factor")



Training data: red: class -1
blue: class 1

k =1
High sensitivity to noise
And outliers

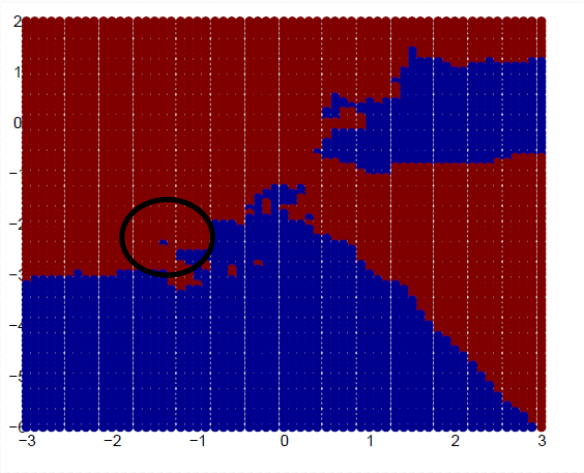


Red: region predicted as -1
Blue: region predicted as 1

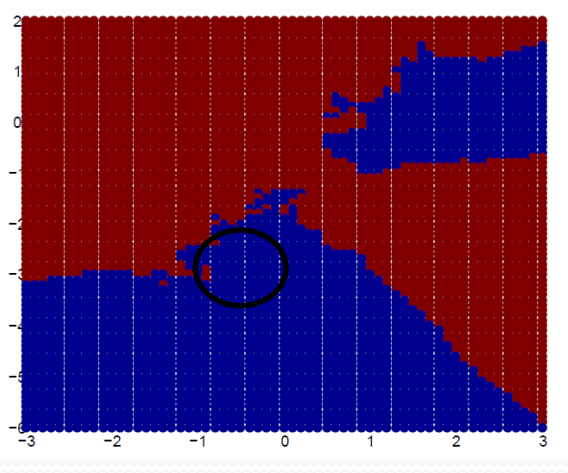
k-NN Algorithm: the Effect of *k*

- *k* plays the role of a capacity control parameter

  (in reality: *k* is a s"smoothing factor")



k=3



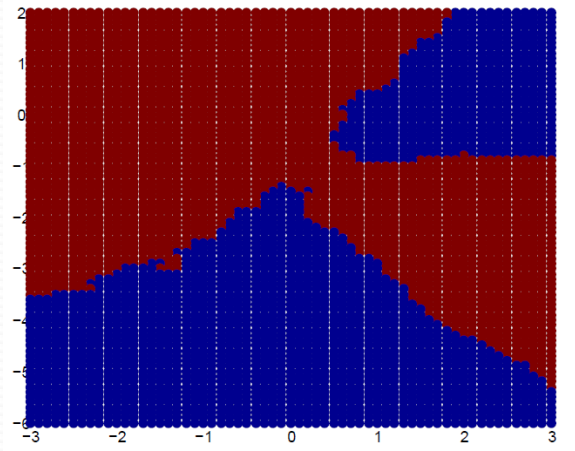
k=3

Black circles: small regions where noise and outliers do not influence the prediction any more

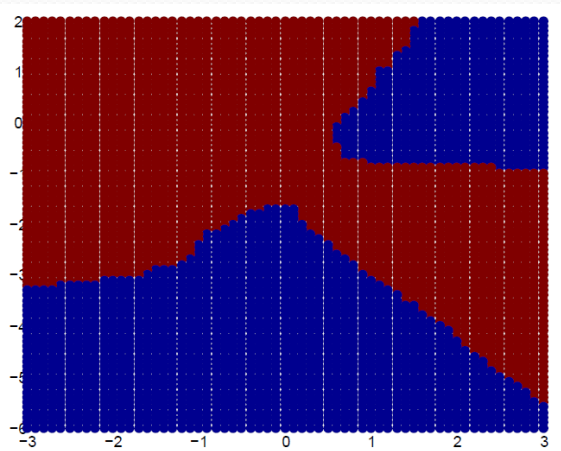
k-NN Algorithm: the Effect of *k*

- *k* plays the role of a capacity control parameter

  (in reality: *k* is a s"smoothing factor")



k=10

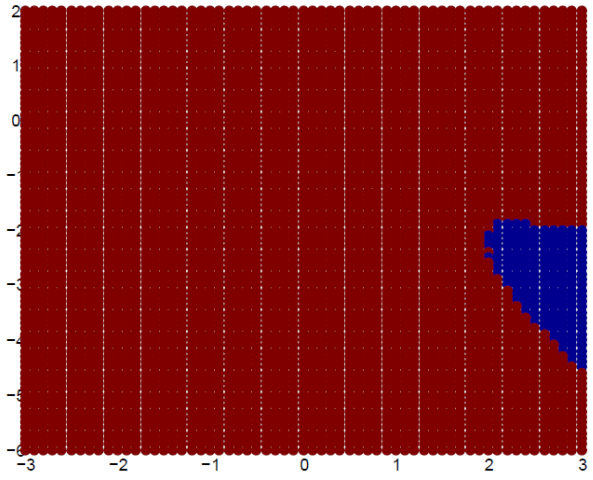


k=30

k-NN Algorithm: the Effect of *k*

- *k* plays the role of a capacity control parameter

(in reality: *k* is a s"smoothing factor")



k=300

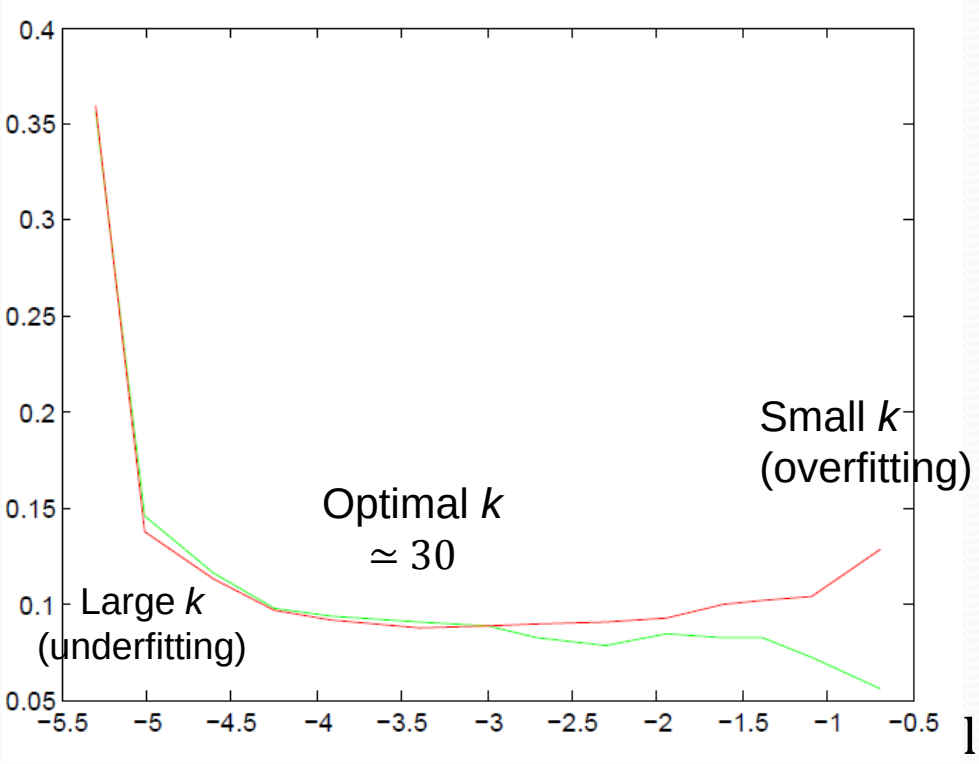
Too many neighbors compared to the number of examples

The majority class (-1) is predicted almost everywhere

k-NN Algorithm: the Effect of *k*

- *k* plays the role of a capacity control parameter

(in reality: *k* is a s"smoothing factor")



Red: generalization error
Green: train error

For the previous data,
500 examples

k-NN Algorithm: Some Insights from Statistical Learning Theory

- k-NN is *universally consistent:*
  - Asymptotic result (using euclidian distance in R^d):

    As the number of (training) examples m grows to infinity,
    if k grows to infinity less fast than m ($k \approx \sqrt{m}$):

    the generalization error of k-nn converges to the best possible error of any prediction function

- Consequences:
  - Lesson 1:

    a distance between instances is all we need to learn
  - Lesson 2:

    k must increase with the number of training examples m

k-NN in Practice 1/2

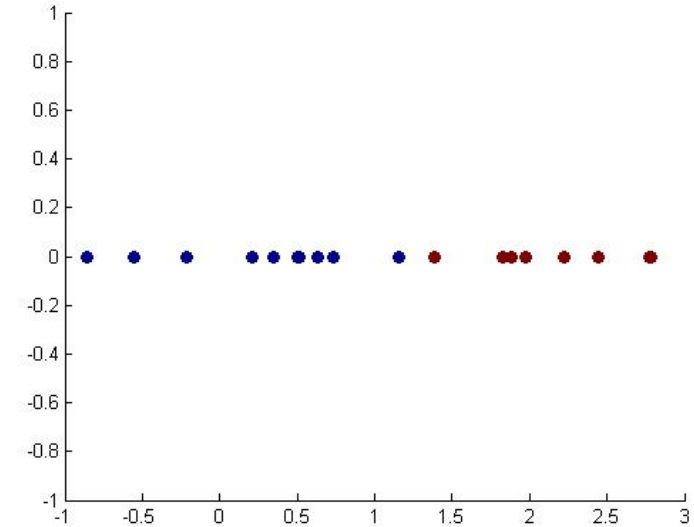
- Choose *k* using a validation set or by cross-validation
  - Theory still gives us an order of magnitude of "good" values of *k*
- Advantages:
  - does not need any prior knowledge about the shape of the decision function
  - Simple to implement, strong asymptotic guarantees
  - Performs well with large training sets and low-dimensional data

k-NN in Practice 2/2

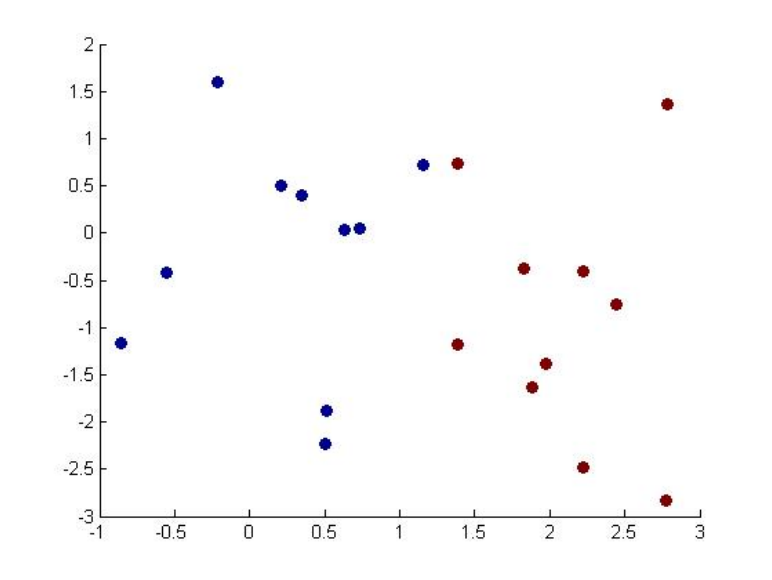
- Computationally costly for prediction:
  - Naive implementation requires $O(m)$ distance computations for each predictions

    (More sophisticated implementations exist using nearest neighbor search algorithms, e.g. KD-trees data structures)
  - In any case: important memory requirements at prediction time
- Needs a large number of examples to adapt to the true decision boundary
  - The algorithm is sensitive to irrelevent features, and to the curse of dimensionality
- The choice of the distance is critical
  - In particular: sensitivity to feature scaling

k-NN: Sensitivity to Irrelevant Features 1/2



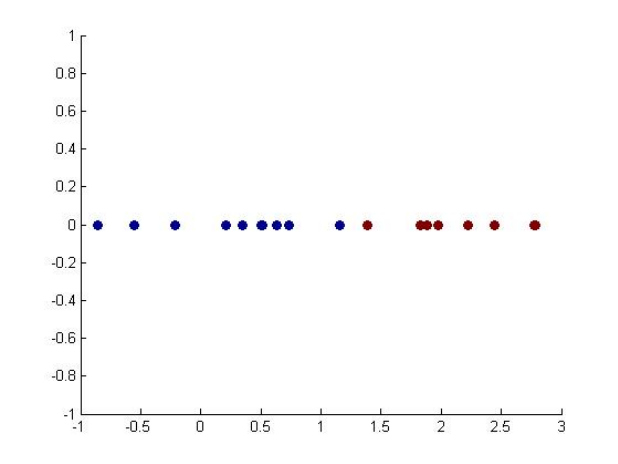
an easy one dimensional classification problem



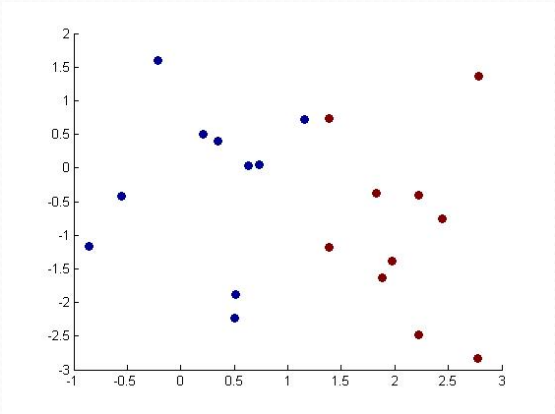
+ irrelevant feature (random *y* values, according to a gaussian distribution)

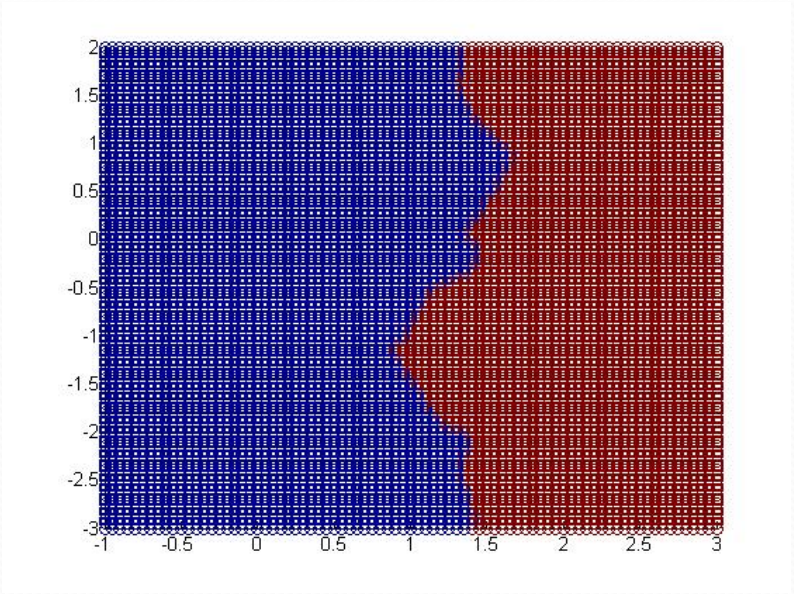
k-NN: Sensitivity to Irrelevant Features 1/2

Original data



+ irrelevant feature





Result of 3-nn

The irrelevant feature affects the shape of the decision boundary

Decision Tree Learning

Decision Trees: an Example

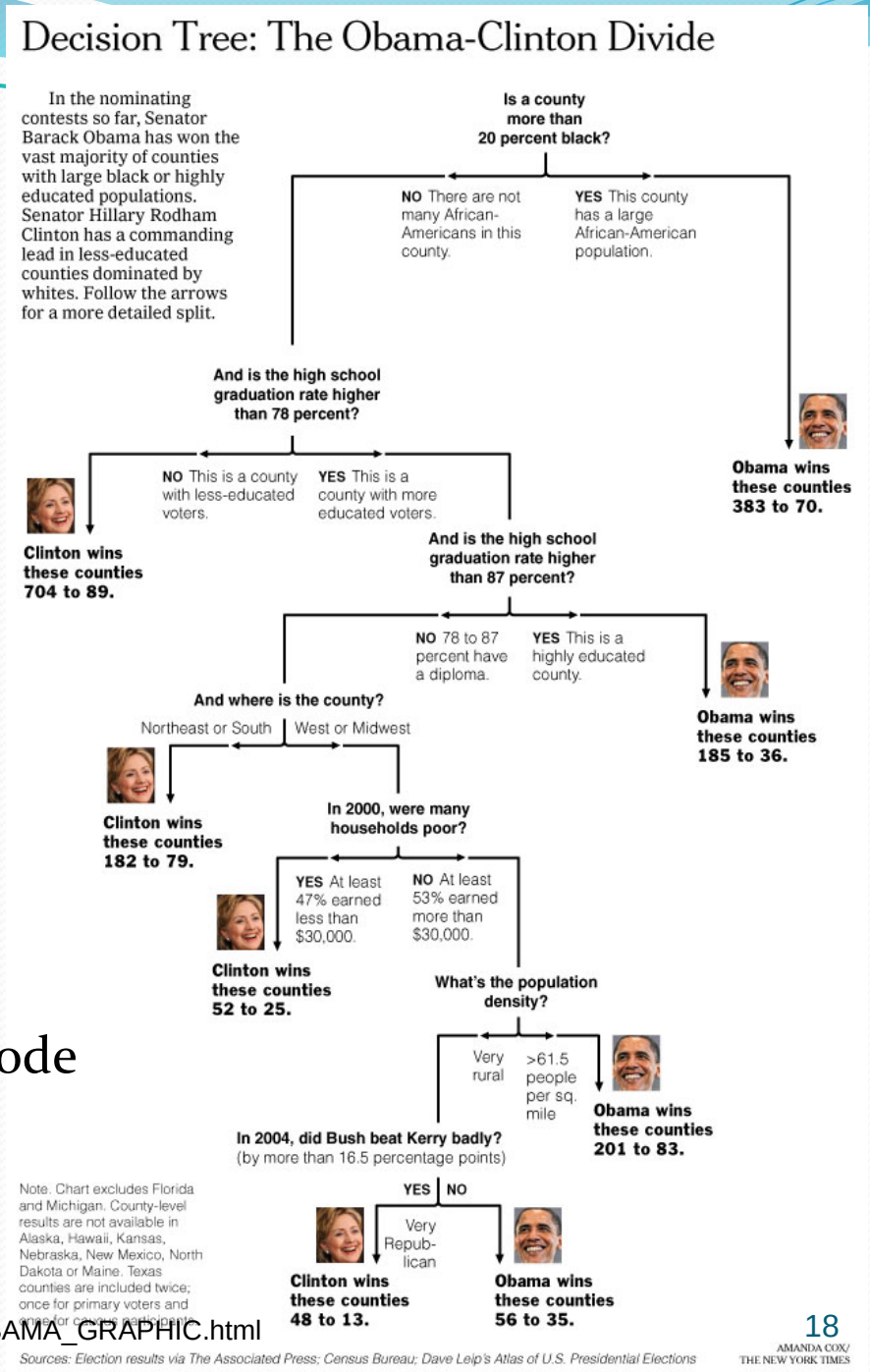
- Decision tree =

  prediction function

- Each node =

  a question

- Prediction algorithm:
  - Start from root
  - Repeat until reaching a leaf
    - Answer the question at the node
    - If answer is yes, go right
    - Else: go left



Source:http://www.nytimes.com/imagepages/2008/04/16/us/20080416_OBAMA_GRAPHIC.html

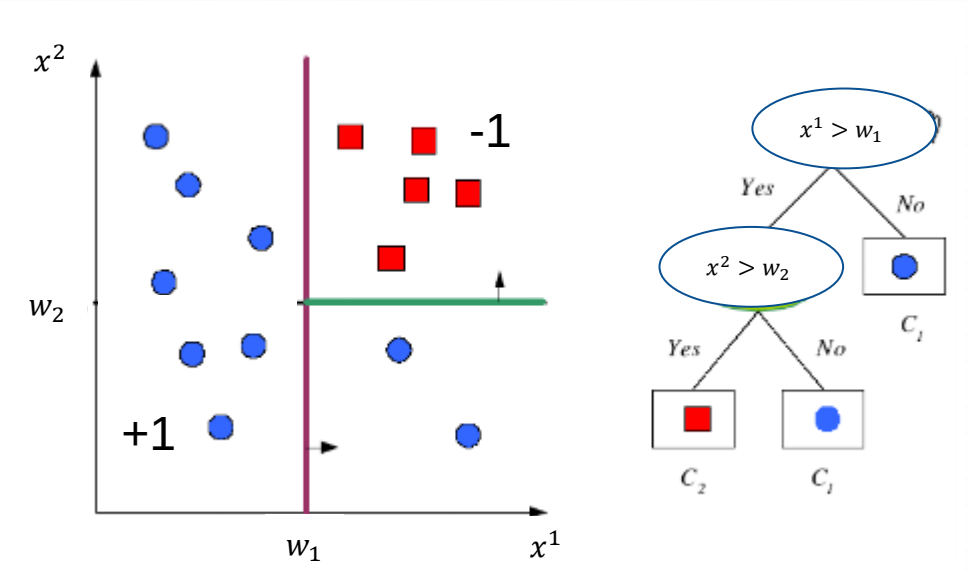
Decision Trees in Machine Learning (1/3)

- Suppose instances are represented by *d*-dimensional vectors
  (trees can also be build using categorical features)
- A question = a feature index + a threshold : "is feature *j* higher than *w*" ?
- DTs decompose the feature space into polytopes of the form

$$x^{j_1} > w_1$$
$$\text{and } x^{j_2} > w_2$$
$$\dots$$
$$\text{and } x^{j_k} > w_k$$

(note: ≤ can also be used instead of >)

- k = length of the path from root to leaf



Source: Decision Tree Learning, Machine Learning course by T. Mitchell

Decision Trees in Machine Learning (2/3)

- Learning decision trees from data is one of the oldest and widely used learning algorithms

- Main advantages:
  - Expressivity:

    Decision trees can approximate any function
  - Interpretability:

    The prediction is the result of a logical AND expression

- They are also used as building blocks in *ensemble methods* (i.e. methods that combine several prediction functions)

Decision Trees in Machine Learning (3/3)

- (A combination of) decision trees is used in the Kinect to classify body parts

source: http://pages.cs.wisc.edu/~dyer/cs540/notes/17_kinect.pdf

- (Combinations of variants of) decision trees are used by Web search engines to generate the Web page ranking

Decision Trees: Learning Algorithm (1/3)

- Learning = "growing" a tree
- There are many decision trees that achieve zero classification errors on a single training set

  (as soon as all instances have distinct feature representations)
- Finding an "optimal" tree is NP-hard

  (e.g. a tree that has the least possible number of leaves and zero classification error on the training set)
- In practice: a greedy, recursive, top-down algorithm to find a tree with few classification errors on the training set

Decision Trees: Learning Algorithm (2/3)

- The recursive function: **split**

  split is recursively applied to each node of the tree

- Input (for the current node):

  a training (sub)set $S_{node} = \{(x_1, y_1), (x_2, y_2), \dots, (x_{m_{node}}, y_{m_{node}})\}$

1. *node* = new node of the tree
2. Find the feature *j* and the threshold *w* that are "best" according to a splitting criterion (see later)
3. Decompose S_{node} into S_{left} and S_{right} as:

   $$S_{left} = \{(x, y) \in S_{node} \mid x^j > w\}$$

   $$S_{right} = \{(x, y) \in S_{node} \mid x^j \leq w\}$$

   Right child of *node* is **split**(S_{right}), left child is **split**(S_{left})
4. return *node*

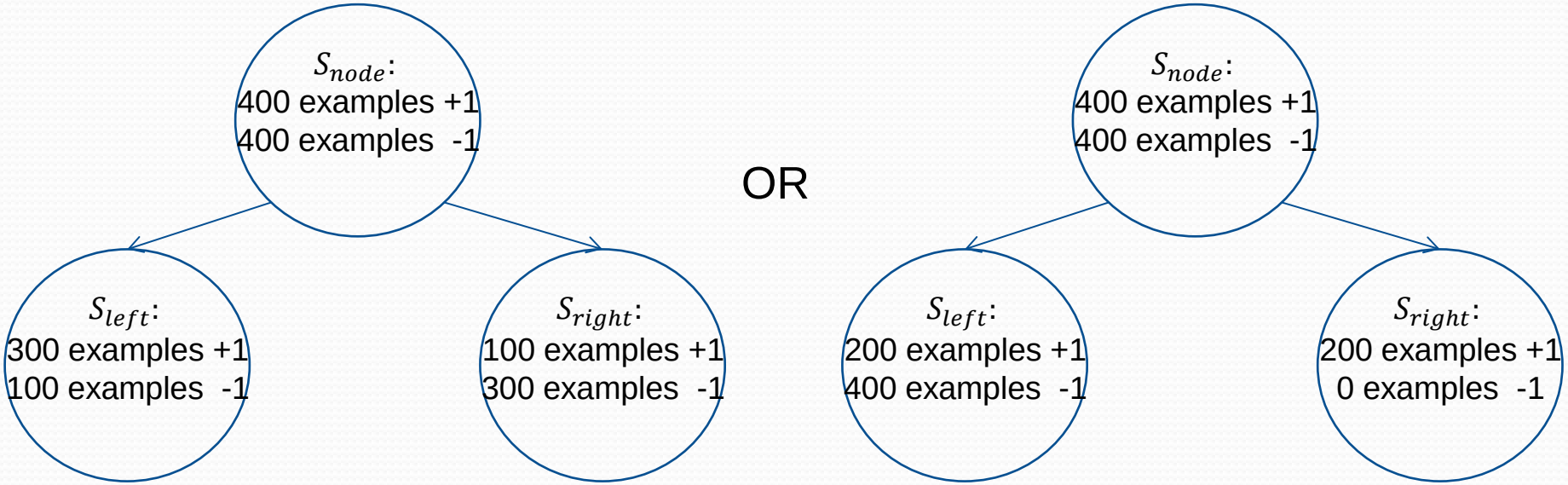
Decision Trees: Learning Algorithm (3/3)

- Function **split** is first applied to the root node, with the whole training data as input
- Usual stopping criteria:
  - If the number of examples at the current node is less than a threshold (e.g. 10)
  - If the number of examples in one of the children nodes is less than a threhold
  - If the depth of the node is greater than a threshold
- The practical rule:

  grow a tree as much as possible (possibly overfitting) and *prune* the tree (i.e. discard subtrees) afterwards

Splitting Rule 1/3

- Selecting the best (feature, threshold) pair:
  - the (feature, threhold) pair that minimizes training error ?



Splitting Rule 2/3

- Choose the (feature, threshold) pair that optimizes an impurity score
  - In practice: the Gini index is widely used
  - Many other impurity scores exist (e.g. information gain)

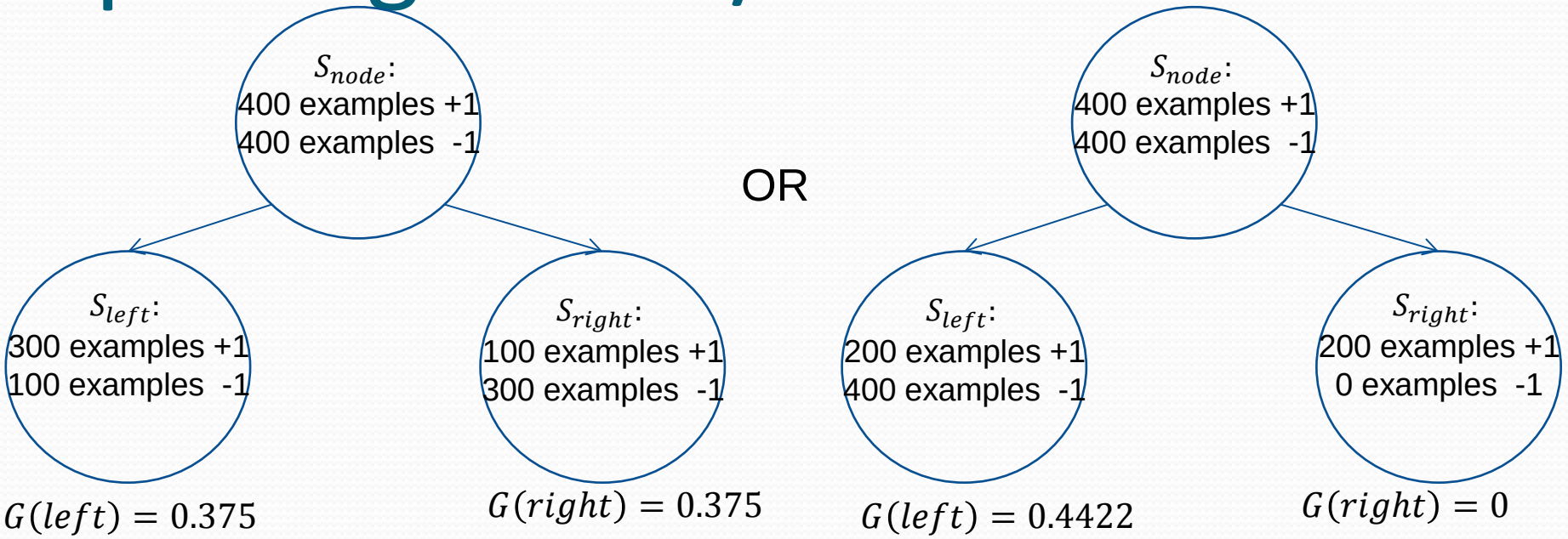
- Gini index of a node (binary classification):

$$G(node) = 1 - p_{-1}(node)^2 - p_{-1}(node)^2$$

Where $p_{-1}(node)$ = proportion of +1 examples in S_{node}

- Impurity of a tree = sum of $G(leaf) * |S_{leaf}|$ over all leaves

Splitting Rule 3/3



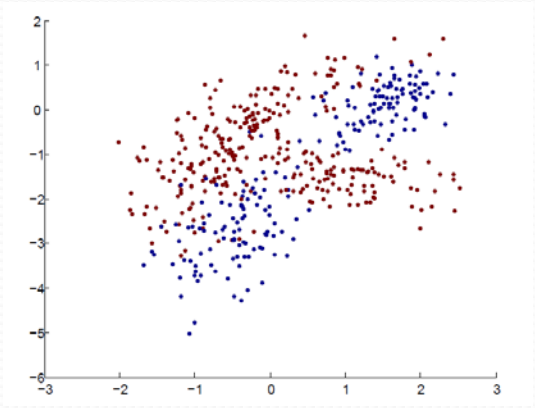
Impurity = $400 * 0.375 + 400 * 0.375 = 300$

Impurity = $600 * 0.4422 + 200 * 0 = 265.32$

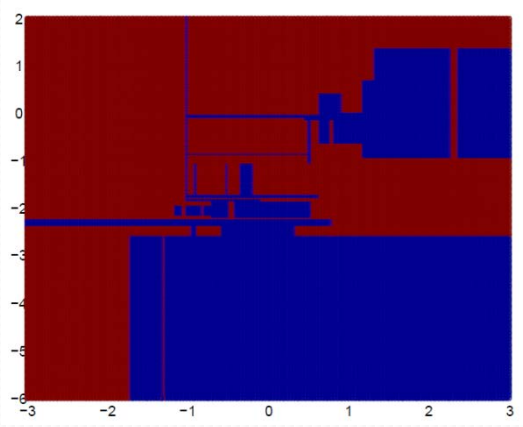
- Choosing the (feature, threshold) pair:
  - Try all possible (feature, threshold) pairs in S_{node}
  - Select the one with minimum impurity

Growing the Tree and Pruning

- Growing a tree can easily lead to overfitting



Training data: red: class -1
blue: class 1



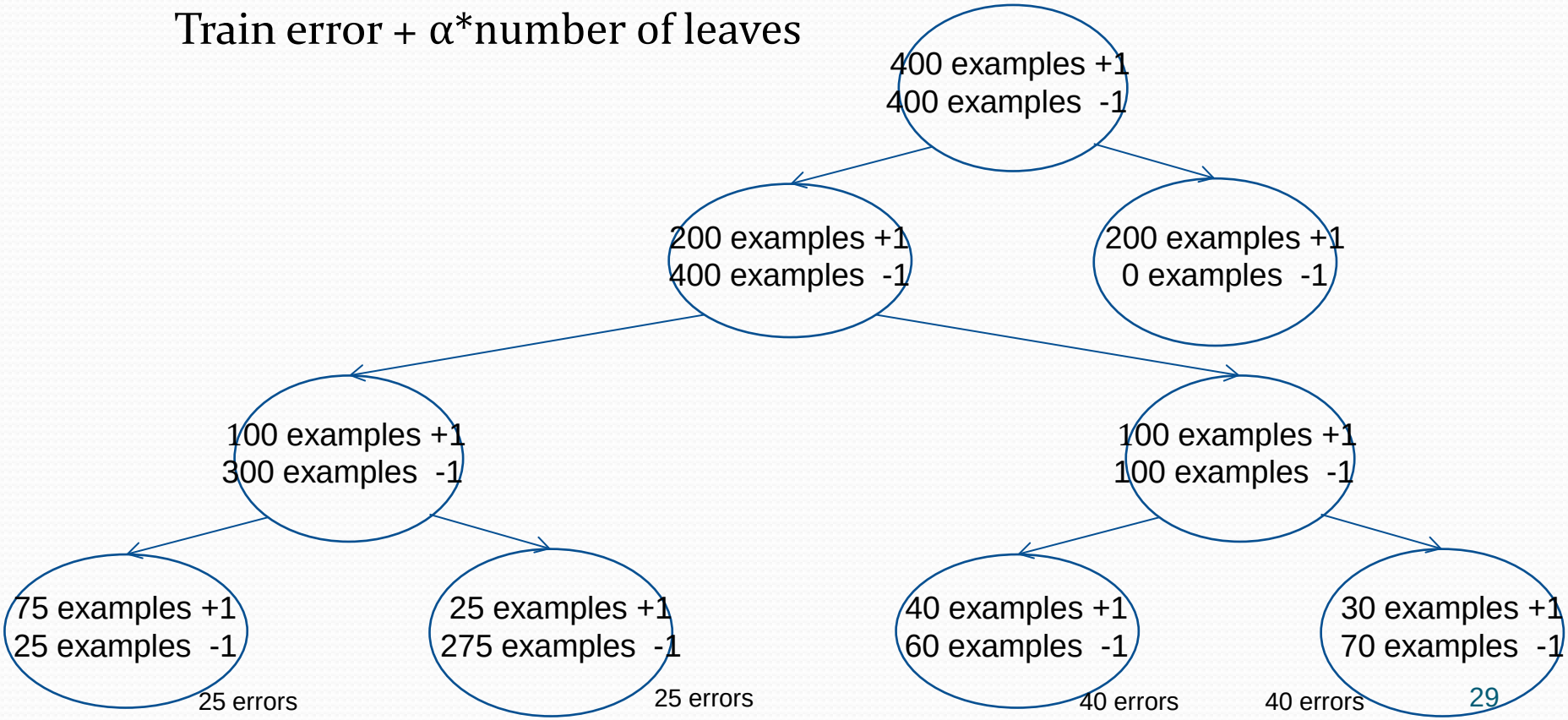
Unpruned tree: red: predict -1
blue: predict 1

- Pruning is the process that implements capacity control in decision trees
- Capacity parameter = number of leaves or number of nodes
- Pruning = replace subtrees by their root node

Pruning and the Optimal Pruning Sequence

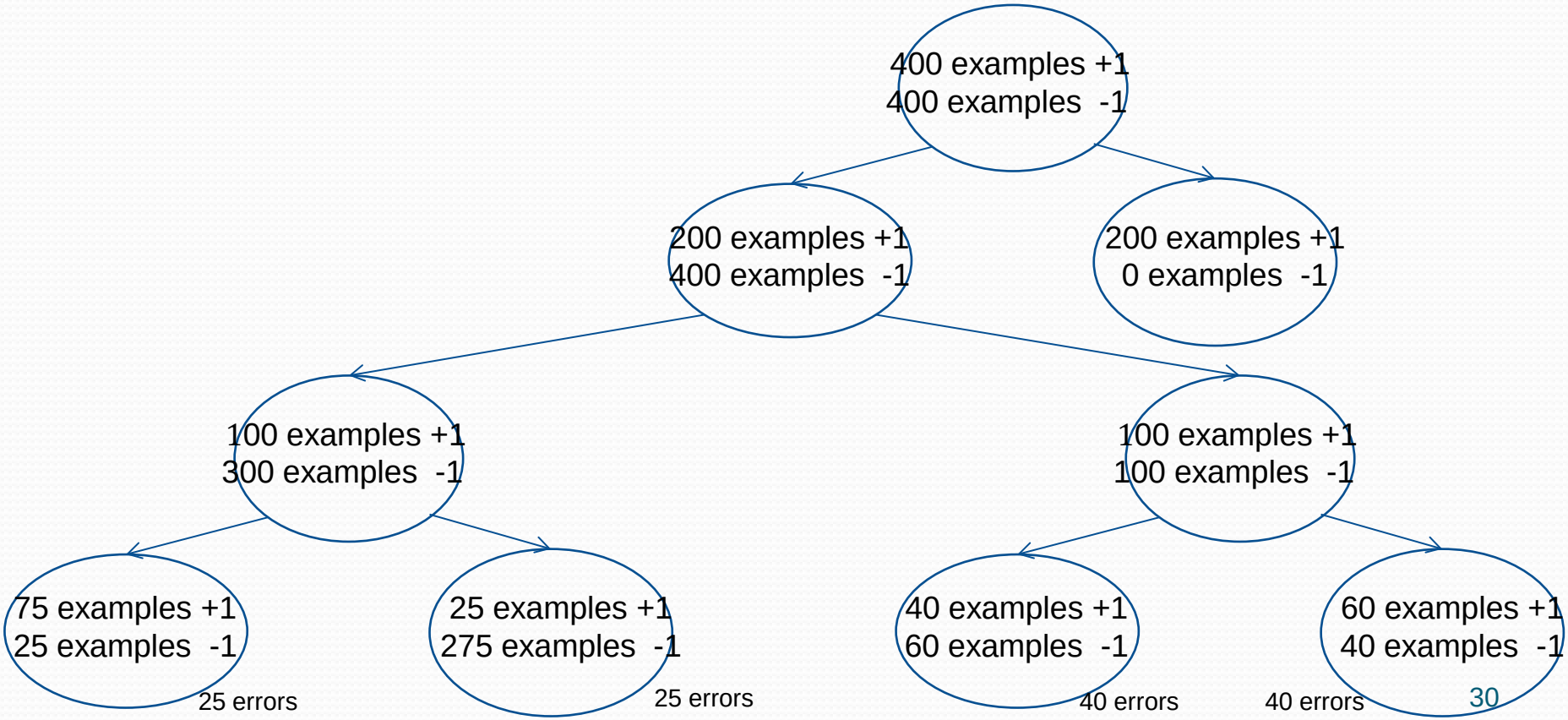
- In the CART algorithm (Breiman et al.):
  - Given a grown tree, subtrees are generated to minimize:

    Train error + α*number of leaves



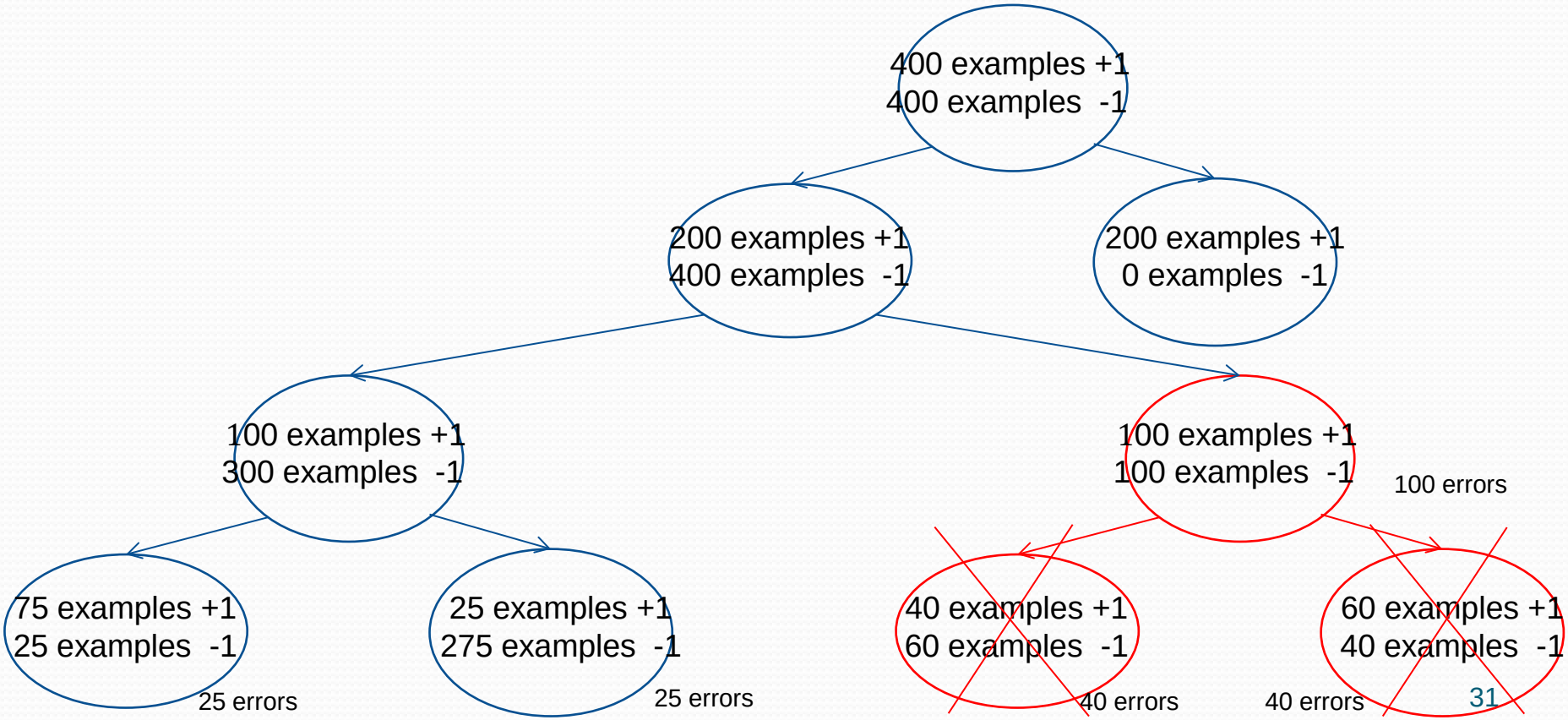
Optimal Pruning Sequence 1/3

- $\alpha=0$: full tree is optimal

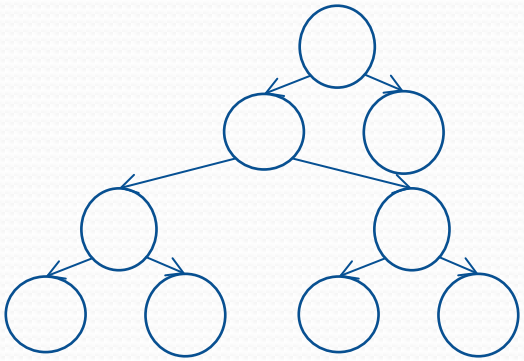


Optimal Pruning Sequence 2/3

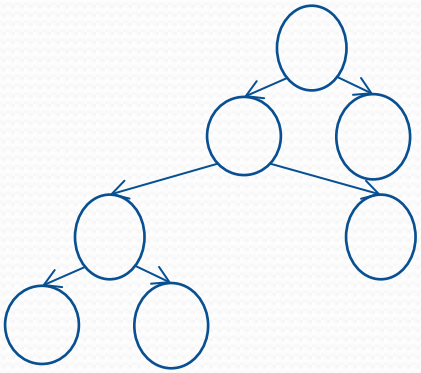
- $0 \leq \alpha \leq 20/800$: full tree is optimal (train error $= 130/800$, nleaves $= 5$)
- $20/800 \leq \alpha$: prune red tree (train error $= 150/800$, nleaves $= 4$)



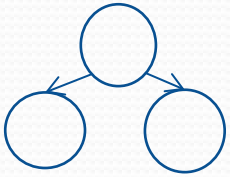
Optimal Pruning Sequence 3/3



Optimal for
$0 \leq \alpha \leq 20/800$



Optimal for
$20/800 \leq \alpha \leq 25/800$



Optimal for
$25/800 \leq \alpha \leq 200/800$

- Choosing the number of leaves of a decision tree on a validation set:
  - Grow a full tree on the training set + build the optimal pruning sequence
  - Select the tree in the sequence that has smallest validation error
- Choosing the best tree by cross-validation:
  - choose the value of α which has best CV error
    + build a new tree on the whole data and take the tree optimal for that α

Decision Trees: Conclusion

- Widely used algorithm in practice
- Advantages:
  - High expressivity (and capacity to overfit),
  - Easy to interpret and visualize
  - Implicit feature selection → little impact of irrelevant features
- Disadvantages:
  - A single tree may not have good performances

    (in practice: use several trees → lecture on ensemble methods)
  - For instance: trees are bad for modeling linear decision functions
- Trees can be built for regression rather than classification
  - Change splitting rule: minimize squared error

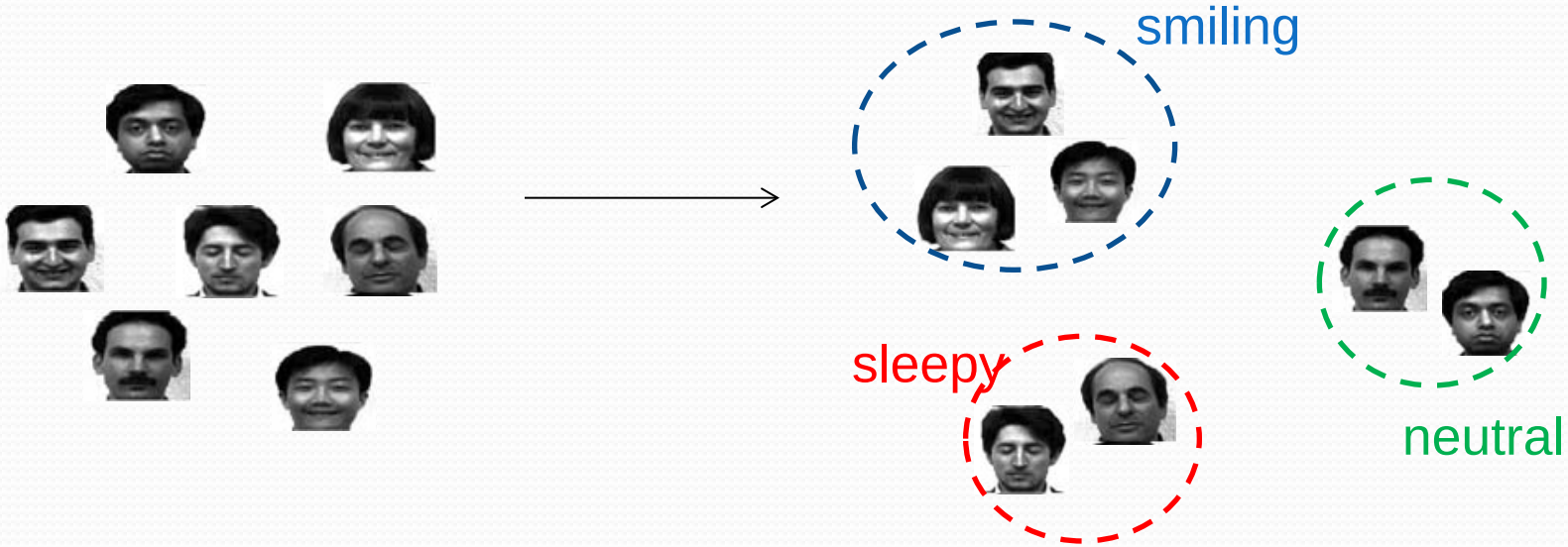
Part II: Clustering

Overview

- Reminder on clustering
- *k*-means clustering
- Hierarchical agglomerative clustering
- Some applications of clustering
- Evaluating a clustering

Reminder on clustering

- Task: discover hidden patterns/classes in the data



- Objects are clustered based on similarity/distance
- Data is *unlabeled*:
  - the desired clusters are not known a priori
  - the interpretation of the clusters is usually left to a human expert

k-means clustering

k-Means Clustering

- Input:
  - dataset $S = \{x_1, x_2, \ldots, x_m\}$
  - Each x_i is in a d-dimensional vector space

  (i.e. each feature must be real-valued, no categorical feature)
  - A distance function between object (typically euclidian distance)
  - Desired number of clusters: k
- Goal:
  - find a *partition* of the data into k clusters
  - Instances in the same cluster should be close to each other
  - Instances in different clusters should be far away from each other

k-Means Clustering (Euclidian Distance)

- Input:
  - dataset $S = \{x_1, x_2, \dots, x_m\}, x_i \in R^d$
  - Desired number of clusters: k
- Output:
  - $c_1, c_2, \dots, c_m$,where $c_i \in \{1, \dots, k\}$: c_i is the cluster to which x_i belongs
  - $M_1, M_2, \dots, M_k$ where $M_j \in R^d$: M_j is the *centroid* of cluster j
- Goal of the algorithm:

$$\begin{array}{c} \text{minimize} \\ c_i \in \{1, \dots, k\} \\ M_j \in R^d \end{array} \quad \sum_{i=1}^{m} \left\| x_i - M_{c_i} \right\|^2$$

k-means clustering (euclidian distance)

- Input:
  - dataset $S = \{x_1, x_2, \dots, x_m\}, x_i \in R^d$
  - Desired number of clusters: k
- Output:
  - $c_1, c_2, \dots, c_m$,where $c_i \in \{1, \dots, k\}$: c_i is the cluster to which x_i belongs
  - $M_1, M_2, \dots, M_k$ where $M_j \in R^d$: M_j is the *centroid* of cluster j
- Goal of the algorithm:

$$\underset{\substack{c_i \in \{1,\dots,k\} \\ M_j \in R^d}}{\text{minimize}} \quad \sum_{i=1}^{m} \left\| x_i - M_{c_i} \right\|^2$$

Centroid of x_i's cluster

Cluster of x_i

k-Means Clustering (Euclidian Distance)

- Goal of the algorithm:

$$\begin{array}{c}\text{minimize} \\ c_i \in \{1, \ldots, k\} \\ M_j \in R^d\end{array} \quad \sum_{i=1}^{m} \left\| x_i - M_{c_i} \right\|^2$$

Centroid of x_i's cluster

Cluster of x_i

- Algorithm

1. Initialize randomly (i.e. randomly generated $M_j \in R^d$)
2. Repeat:
   1. For each $i \in \{1, \ldots, m\}$: $c_i = argmin_{j \in \{1,\ldots,k\}} \left\| x_i - M_j \right\|$ (assign x_i to the cluster with nearest centroid)
   2. For each $j \in \{1, \ldots, k\}$: $M_j = \frac{1}{m_j} \sum_{i:c_i=j} x_i$

m_j: number of instances in cluster j

Sum over all instances in cluster j

k-Means Clustering (Euclidian Distance)

- Goal of the algorithm:

$$\begin{array}{c}\text{minimize} \\ c_i \in \{1, \dots, k\} \\ M_j \in R^d\end{array} \quad \sum_{i=1}^{m} \left\| x_i - M_{c_i} \right\|^2$$

Centroid of x_i's cluster

Cluster of x_i

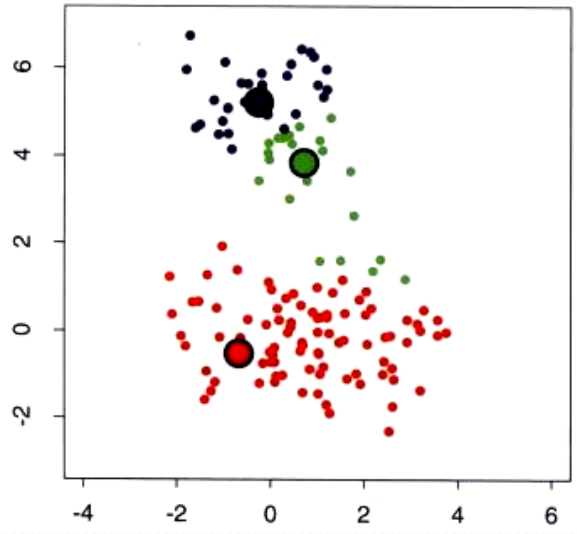
- Algorithm

1. Initialize randomly (i.e. randomly generated $M_j \in R^d$)
2. Repeat (until no change from one iteration to the other):
   1. For each $i \in \{1, \dots, m\}$: $c_i = argmin_{j \in \{1,\dots,k\}} \left\| x_i - M_j \right\|$ (assign x_i to the cluster with nearest centroid)
   2. For each $j \in \{1, \dots, k\}$: $M_j = \frac{1}{m_j} \sum_{i:c_i=j} x_i$

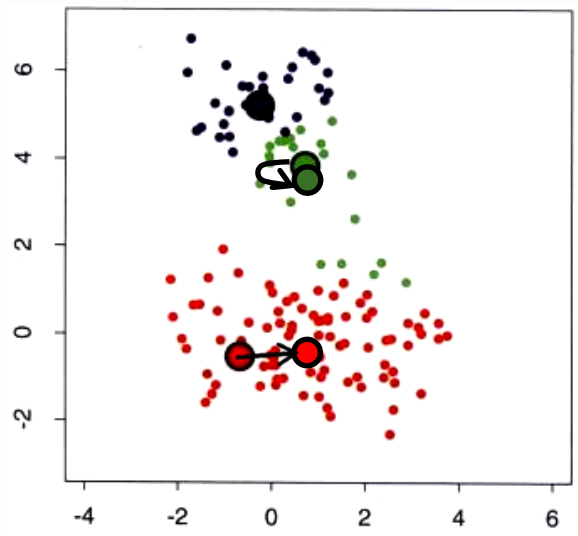
m_j: number of instances in cluster j

Sum over all instances in cluster j

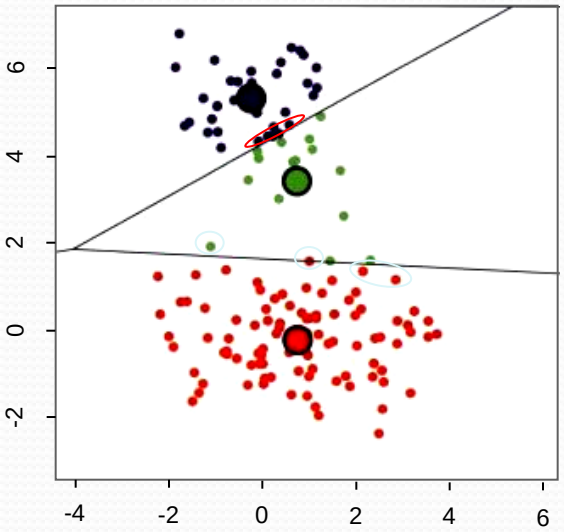
k-Means Clustering (Euclidian Distance)



Random initialization

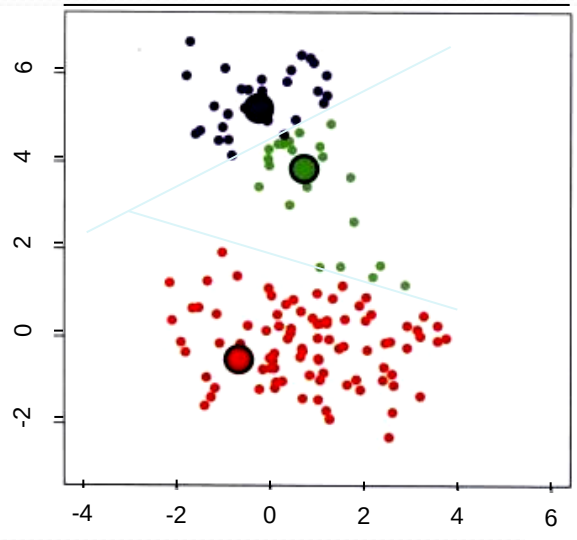


Step 1: Recompute centers

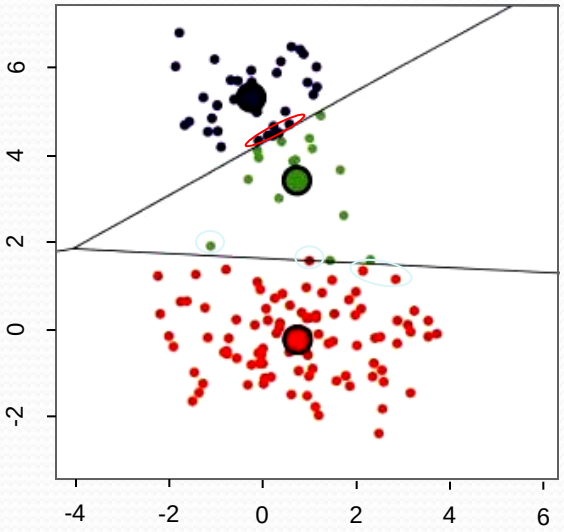


Step 2: reassign instances

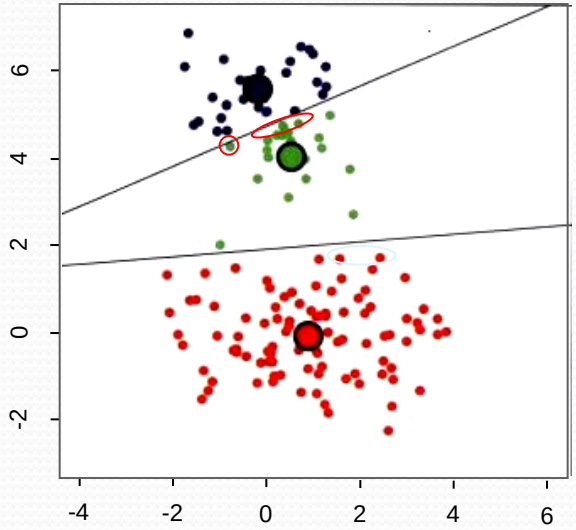
k-Means Clustering (Euclidian Distance)



Random initialization



1st itération:
Recompute center & reassign



20th itération: end

k-Means Clustering: Convergence (1/2)

- Goal of the algorithm:

$$\begin{array}{c} \text{minimize} \\ c_i \in \{1, \ldots, k\} \\ M_j \in R^d \end{array} \quad \sum_{i=1}^{m} \left\| x_i - M_{c_i} \right\|^2$$

- k-means performs *alternating optimization*

- Repeat (until no change from one iteration to the other):
  1. For each $i \in \{1, \ldots, m\}$: $c_i = argmin_{j \in \{1,\ldots,k\}} \left\| x_i - M_j \right\|$
     minimize over c_i when the M_js are fixed
  2. For each $j \in \{1, \ldots, k\}$: $M_j = \frac{1}{m_j} \sum_{i:c_{i=j}} x_i$
     minimize over M_j when the c_is are fixed

k-Means Clustering: Convergence (2/2)

- Goal of the algorithm:

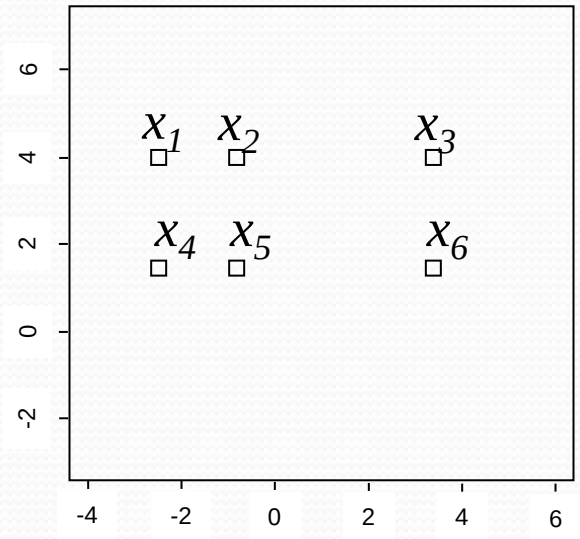
$$\begin{array}{c} \text{minimize} \\ c_i \in \{1, \ldots, k\} \\ M_j \in R^d \end{array} \quad \sum_{i=1}^{m} \left\| x_i - M_{c_i} \right\|^2$$

- k-means performs *alternating optimization*
  - Objective function decreases at each step
  - Objective function bounded below (by 0)
  - Conclusion: the objective function converges

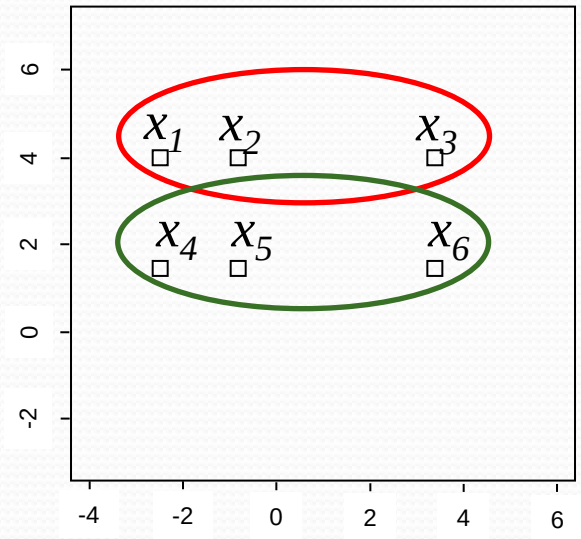
- The algorithm stops after a finite number of iteration:
  - There are k^m possible values of the c_is
  - Each possible assignment can only be visited one time

k-Means Clustering: Convergence to a Local Minimum

- k-means converges to a *local* minimum
  - the final clustering may depend on the initialization



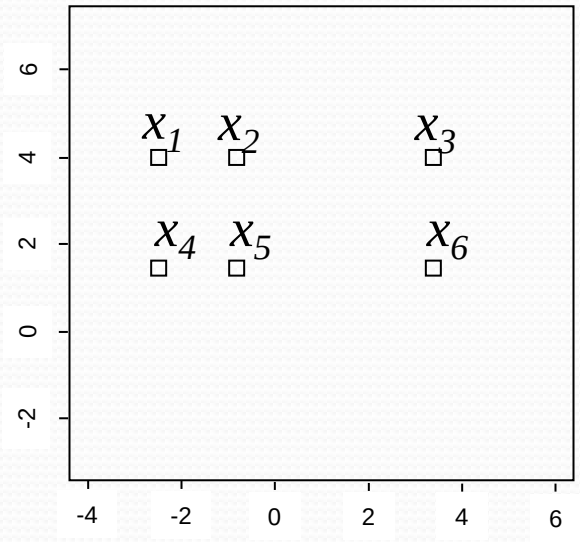
Initialization 1:
$M_1 = x_2$ and $M_2 = x_5$



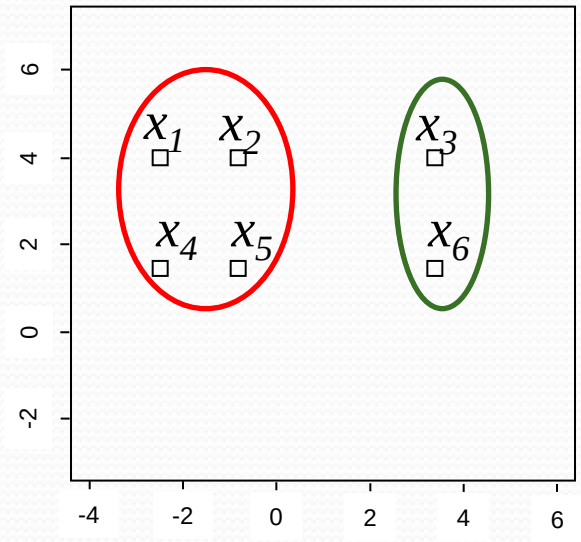
Final clustering

k-Means Clustering: Convergence to a Local Minimum

- k-means converges to a *local* minimum
  - the final clustering may depend on the initialization



Initialization 2:
$M_1 = x_2$ and $M_2 = x_3$



Final clustering

k-Means Clustering: Initialization

- Initializing *k*-means:
  - Randomly generated vectors in the range of the data
  - Centroids computed based on a random assignment of the instances
  - Using another clustering algorithm (i.e. hierarchical agglomerative clustering) applied to a small subset of the data

- Random restarts:
  - Restart the algorithm with several random initializations
  - Take the solution with smaller objective value

k-means clustering: choosing the right number of clusters

- Setting k is related to the problem of evaluation
  - Manual inspection, indirect evaluation (see end of the lecture)
- Another approach:

  trading-off model complexity and objective function value
  - Goal similar to penalty approaches to capacity control
  - Example: the Bayesian Information Criterion (BIC) heuristic:

  Run the algorithm for different values of k and choose the one that minimizes:

$$\sum_{i=1}^{m} \left\| x_i - M_{c_i} \right\|^2 + \frac{k*d}{2} \ln(m)$$

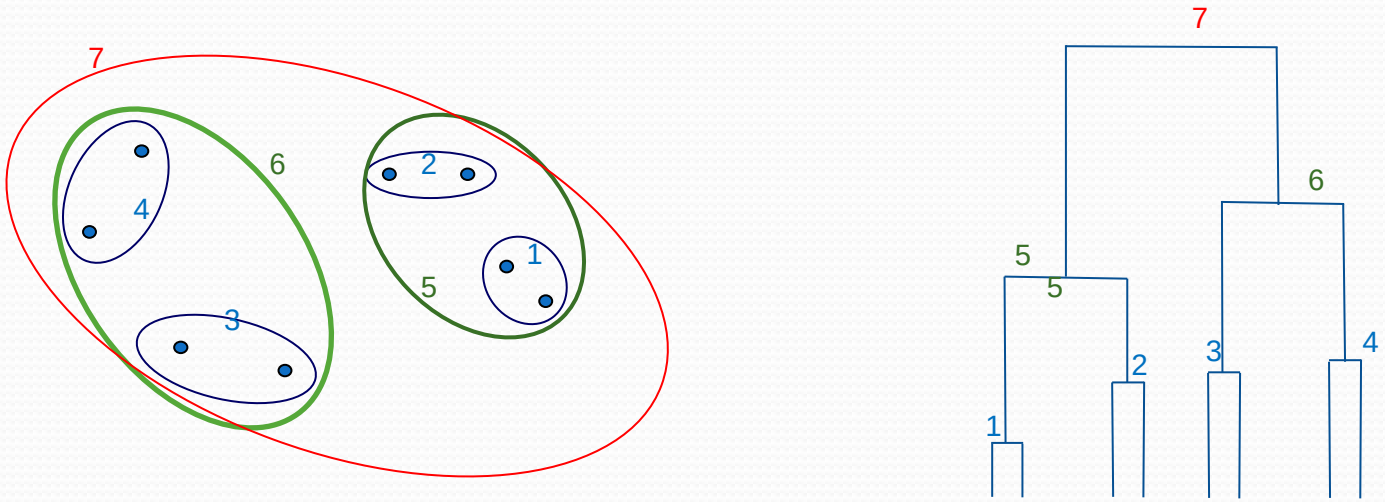
  - BIC inspired by a probabilistic interpretation of k-means

    (the objective function is the likelihood of a model where data in each cluster is generated by a gaussian distribution)
  - Other criteria exist,e.g. Akaike Information Criterion, Silhouette, etc.

HAC:
Hierarchical Agglomerative Clustering

Hierarchical Clustering

- Goal: organize data into a tree-like hierarchy
  - Root of the tree = all the data
  - Bottom of the tree: instances
    (considered as clusters containing a single instance)
  - Each node merges the two "most similar" clusters



Hierarchical Agglomerative Clustering

- HAC: a bottom-up algorithm for hierarchical clustering
- Input:
  - a dataset $S = \{x_1, x_2, \ldots, x_m\}$ of arbitrary instances
  - A matrix D of pairwise "distances"

    ($D_{i,j}$ = distance between x_i and x_j)
  - A dissimilarity function between 2 clusters of instances
- Algorithm:
  - Initialize: sets of clusters C containing m single-instance clusters
  - Repeat while C is not empty:
  1. Merge the two less dissimilar clusters, remove them from C
  2. Add the new (merged) clusters to C
- Output: all clusters found in the process, organized in a tree

Hierarchical Agglomerative Clustering: dissimilarity functions

- Let A and B be two clusters
- "single-linkage" HAC:

$$d(A,B) = min_{i \in A, j \in B} \quad D_{i,j}$$

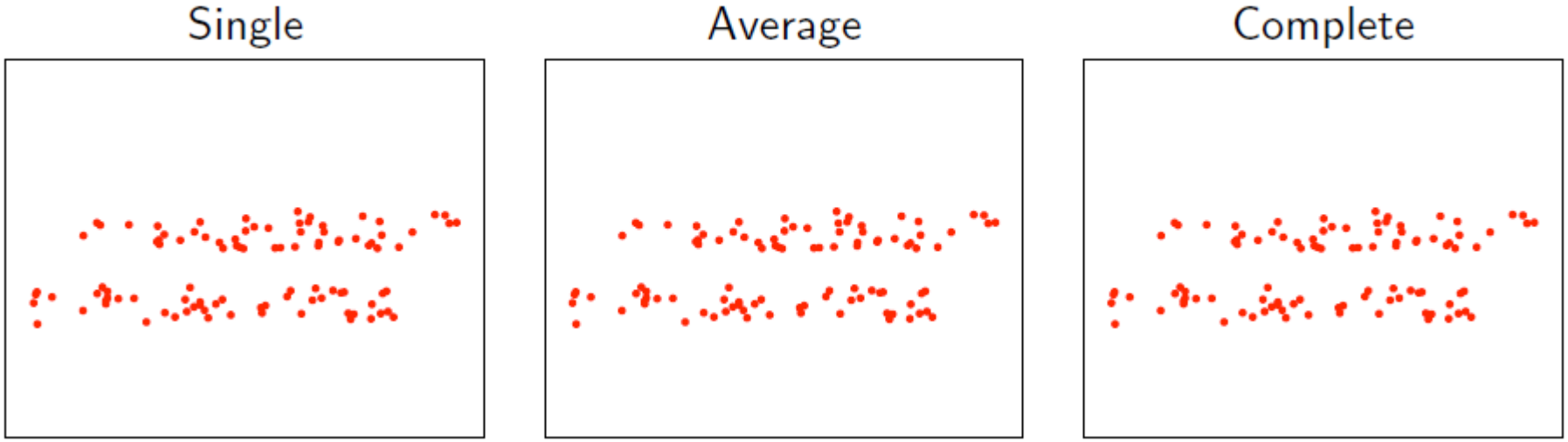
- "complete-linkage" HAC:

$$d(A,B) = max_{i \in A, j \in B} \quad D_{i,j}$$

- "group-average" HAC:

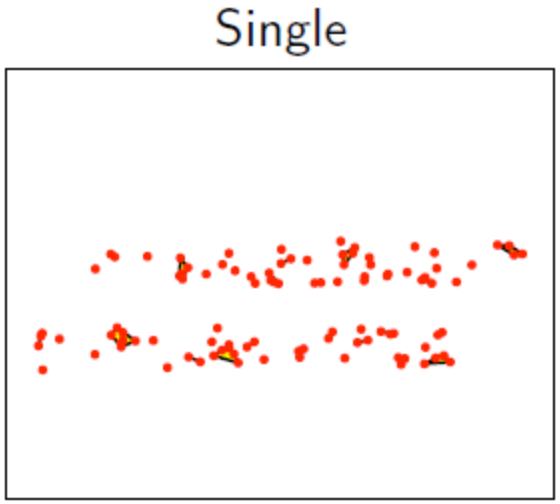
$$d(A,B) = \frac{1}{|A||B|} \sum_{i \in A, j \in B} D_{i,j}$$

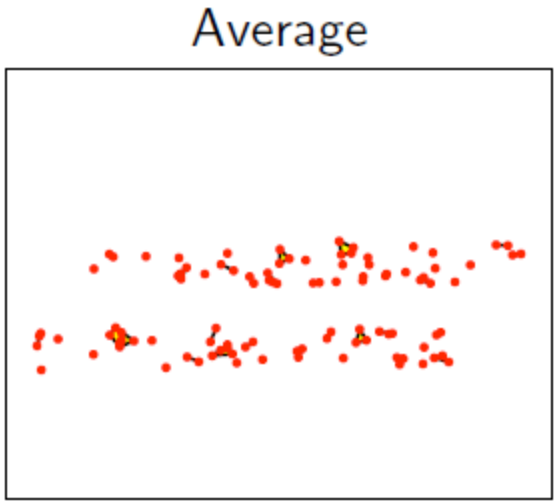
HAC: Example (initial data)

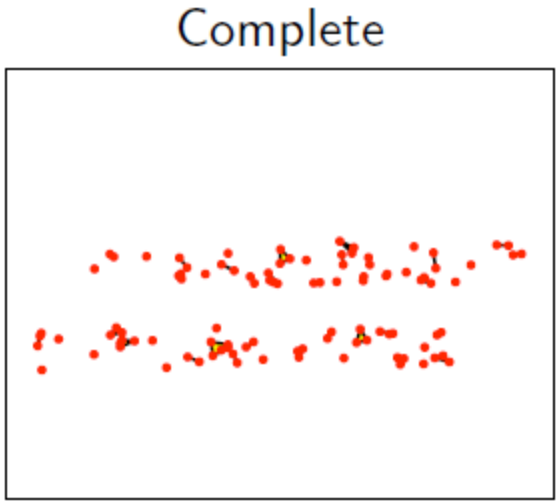


source: http://www.cs.mcgill.ca/~dprecup/courses/ML/Lectures/ml-lecture15.pdf

HAC: Example (Iteration 50)

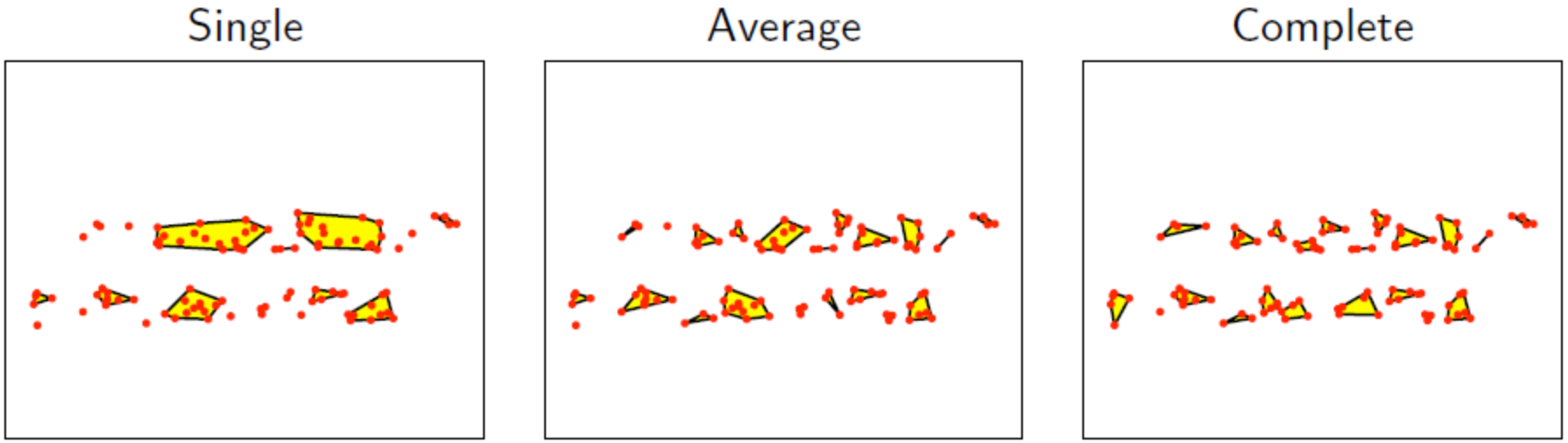






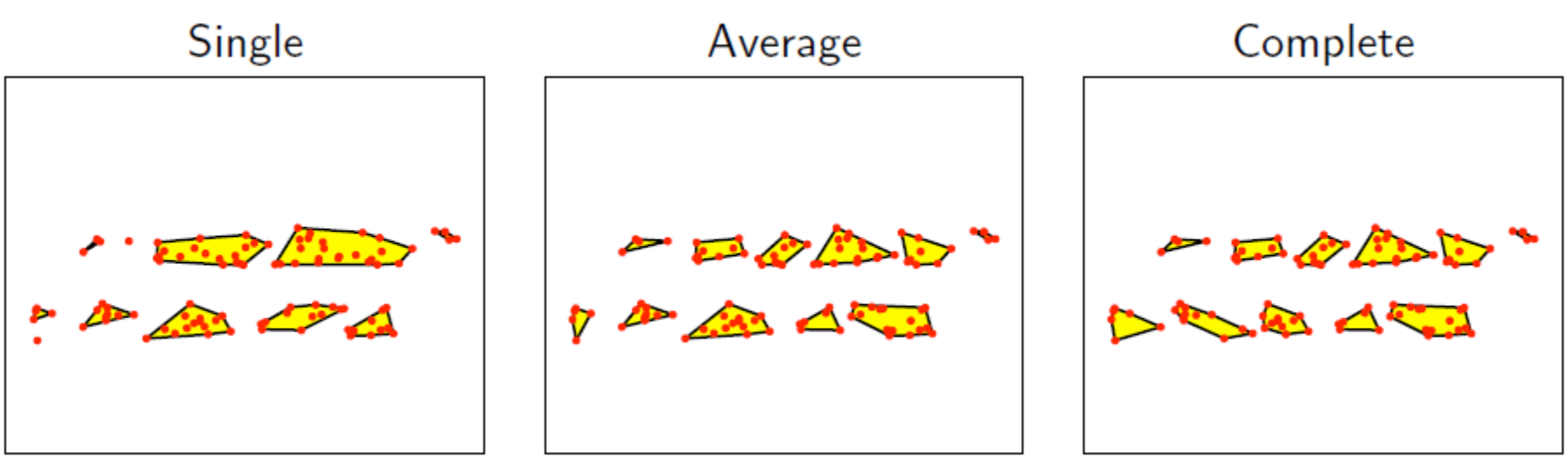
source: http://www.cs.mcgill.ca/~dprecup/courses/ML/Lectures/ml-lecture15.pdf

HAC: Example (Iteration 80)



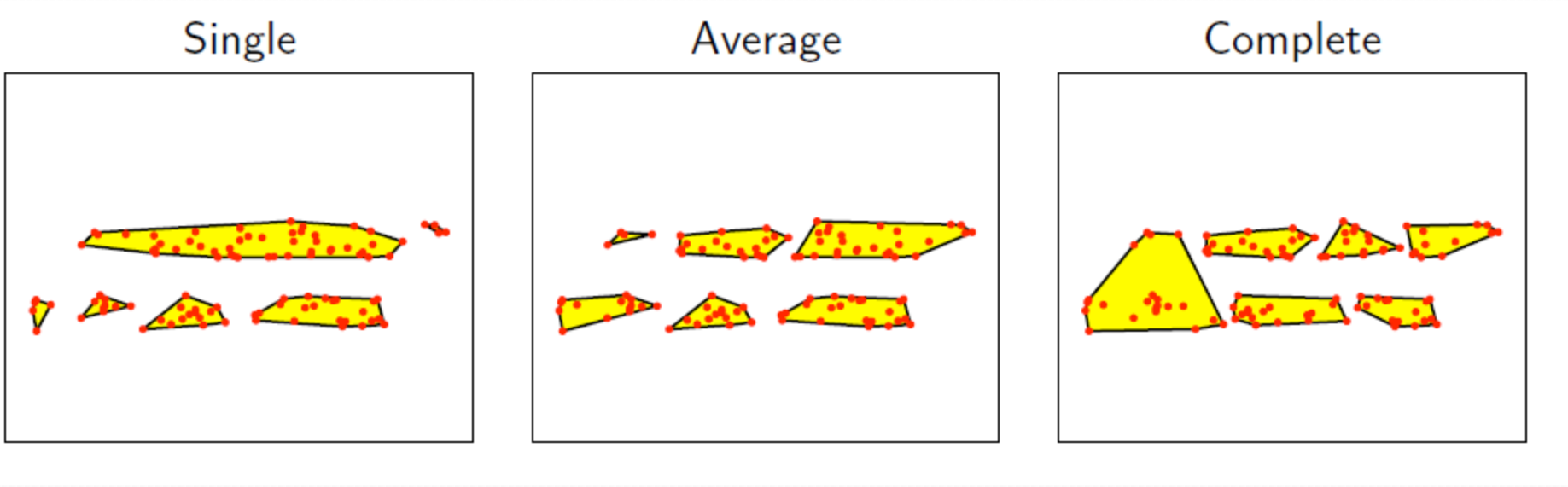
source: http://www.cs.mcgill.ca/~dprecup/courses/ML/Lectures/ml-lecture15.pdf

HAC: Example (Iteration 90)



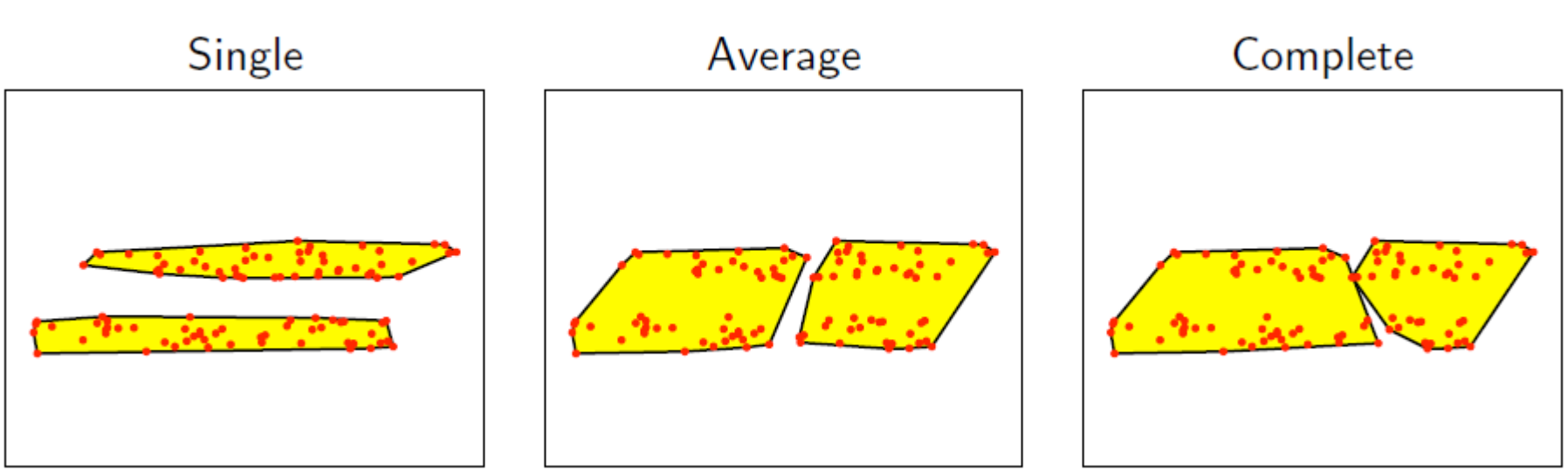
source: http://www.cs.mcgill.ca/~dprecup/courses/ML/Lectures/ml-lecture15.pdf

HAC: Example (Iteration 95)



source: http://www.cs.mcgill.ca/~dprecup/courses/ML/Lectures/ml-lecture15.pdf

HAC: Example (Iteration 99)



Single-linkage:

Clusters propagate to nearest neighbors

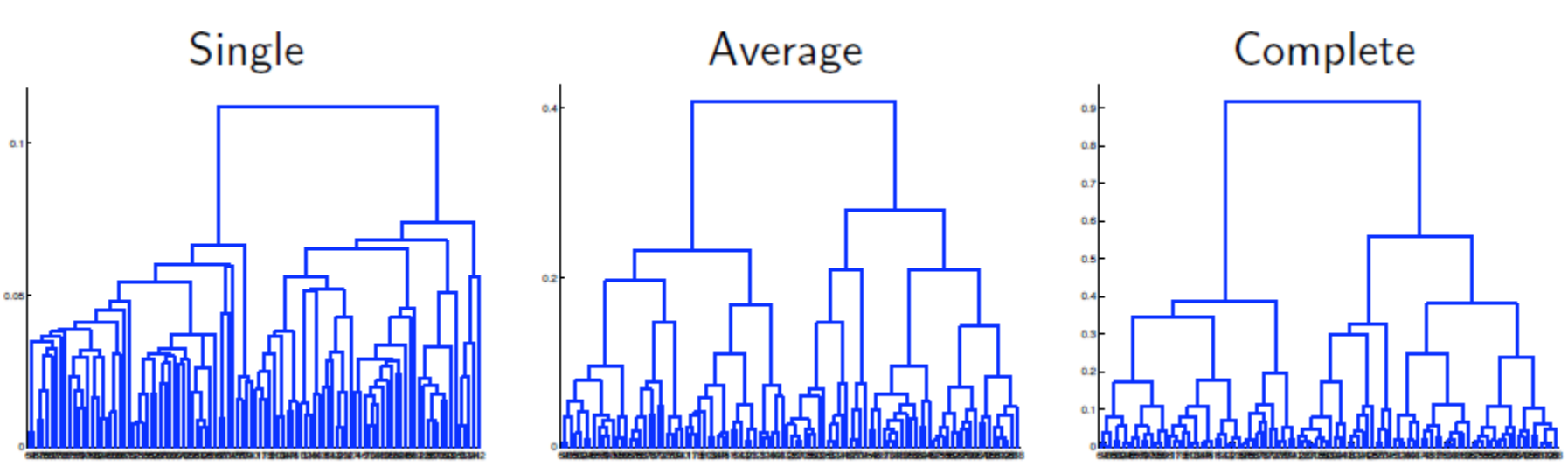
Complete-linkage:

Clusters correspond to denser regions

source: http://www.cs.mcgill.ca/~dprecup/courses/ML/Lectures/ml-lecture15.pdf

HAC: the Dendogram

- Dendogram: tree-structure representing all clusters
- The *y*-axis of the dendogram is the dissimilarity value
- Thresholding the dendogram allows to choose the desired number of clusters

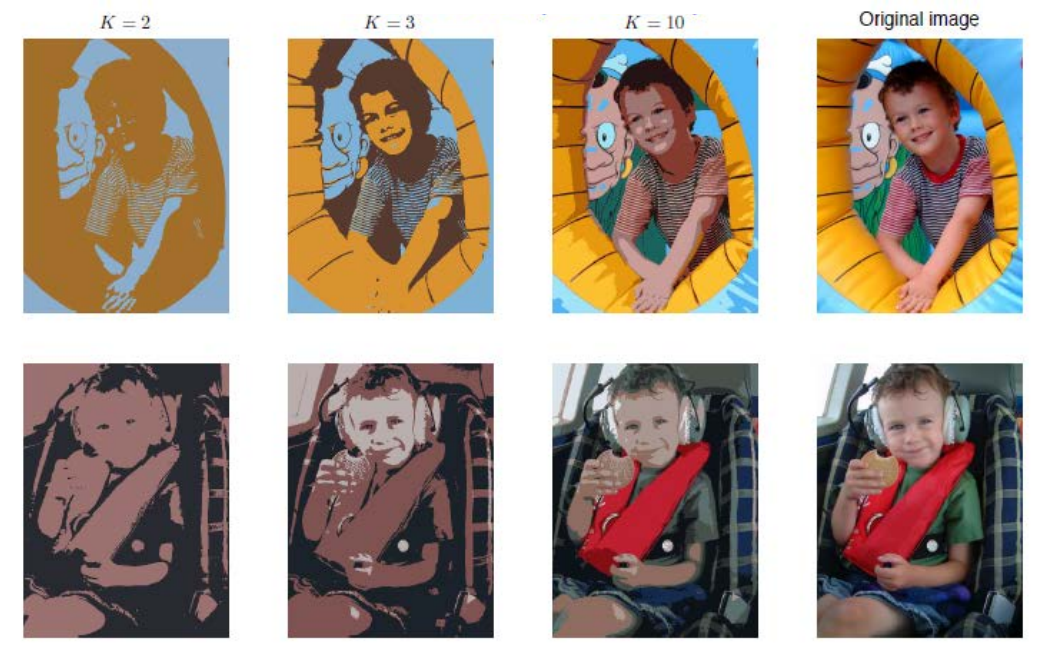


source: http://www.cs.mcgill.ca/~dprecup/courses/ML/Lectures/ml-lecture15.pdf

Applications of Clustering

Application 1: Vector Quantization

- Image compression (example from a book by C. Bishop):
  - Pixels on 24 bits → a small number of *representative colors*
  - Representative colors are obtained by *k*-means clustering
    (cluster the pixels based on their RGB values)



Application 1 (bis): Creating Codewords

- Supervised learning (classification) problem:

Grass
(class 1)

VS

Sky
(class 2)

- Colors should be good features:
  - Represent images on a color histogram
    (one color = one feature, value = number of pixels of that color)
  - Difficulty: colors on 24 bits $\Rightarrow 2^{24}$ features, no "similar" colors

- Solution: cluster the pixels of the whole dataset to find a few (e.g. 1000) "typical" colors to reduce the dimensionality of the problem

Application 2: Find Groups of Similar Items

- Example from "using co-clustering for predicting movie ratings in NetFlix" (by T. Huynh and D. Vu)

(note: uses co-clustering, more involved algorithms than *k*-means/HAC to cluster both the rows and columns of a matrix)

Movie clusters obtained from user ratings

(grouping movies that received similar ratings from similar users)

MovieID	Title
538	Scarface
3962	GoodFellas
1134	Casino
161	North by Northwest

MovieID	Title
1387	Talk to Her
11	Immortal Beloved
210	High Fidelity
222	Chasing Amy
12205	They Shoot Horses

MovieID	Title
1692	Lord of the Rings: The Fellowship of the Ring
4873	Lord of the Rings: The Two Towers
9820	Lord of the Rings: The Return of the King
3857	Star Wars: Episode V: The Empire Strikes Back
6634	Star Wars: Episode VI: Return of the Jedi

Evaluating Clusterings

- Direct evaluation by a human expert
- Depending on the application, an *indirect* evaluation may be possible:
  - If the clusters are used as a preprocessing of a supervised learning algorithm → final performance
  - If we have labels, a classification error can be computed

    (assign to each cluster member the class that appears most often in the cluster)

Clustering: Conclusion

- *k*-means and HAC are widely used clustering algorithms
  - Many other algorithms and variants:
    - Xmeans instead of *k*-means (includes BIC criterion)
    - Maximize likelihood of a mixture of gaussian models
    - Top-down hierarchical clustering
- Clustering algorithms can be used to:
  - Build prototypes, summarize/compress/visualize data
  - preprocess data for a supervised learning algorithm
- Evaluation of clustering is an art
  - Indirect evaluations are sometimes possible