



Instance-Based Learning

- Rote Learning
- k Nearest-Neighbor Classification
 - Prediction, Weighted Prediction
 - choosing k
 - feature weighting (RELIEF)
 - instance weighting (PEBLS)
 - efficiency
 - kD-trees

- IBL and Rule Learning
 - NEAR: Nearest Nested Hyper-Rectangles
 - RISE

Acknowledgements:

Some slides adapted from

- Tom Mitchell
- Eibe Frank & Ian Witten
- Kan, Steinbach, Kumar
- Ricardo Gutierrez-Osuna
- Gunter Grieser



Instance Based Classifiers

- No model is learned
 - The stored training instances themselves represent the knowledge
 - Training instances are searched for instance that most closely resembles new instance
- *lazy learning*
- Examples:
 - Rote-learner
 - Memorizes entire training data and performs classification only if attributes of record match one of the training examples exactly



Rote Learning

Day	Temperature	Outlook	Humidity	Windy	Play Golf?
07-05	hot	sunny	high	false	no
07-06	hot	sunny	high	true	no
07-07	hot	overcast	high	false	yes
07-09	cool	rain	normal	false	yes
07-10	cool	overcast	normal	true	yes
07-12	mild	sunny	high	false	no
07-14	cool	sunny	normal	false	yes
07-15	mild	rain	normal	false	yes
07-20	mild	sunny	normal	true	yes
07-21	mild	overcast	high	true	yes
07-22	hot	overcast	normal	false	yes
07-23	mild	rain	high	true	no
07-26	cool	rain	normal	true	no
07-30	mild	rain	high	false	yes

today	cool	sunny	normal	false	yes
-------	------	-------	--------	-------	-----



Instance Based Classifiers

- No model is learned
 - The stored training instances themselves represent the knowledge
 - Training instances are searched for instance that most closely resembles new instance
- *lazy learning*
- Examples:
 - Rote-learner
 - Memorizes entire training data and performs classification only if attributes of record match one of the training examples exactly
 - Nearest-neighbor classifier
 - Uses k “closest” points (nearest neighbors) for performing classification



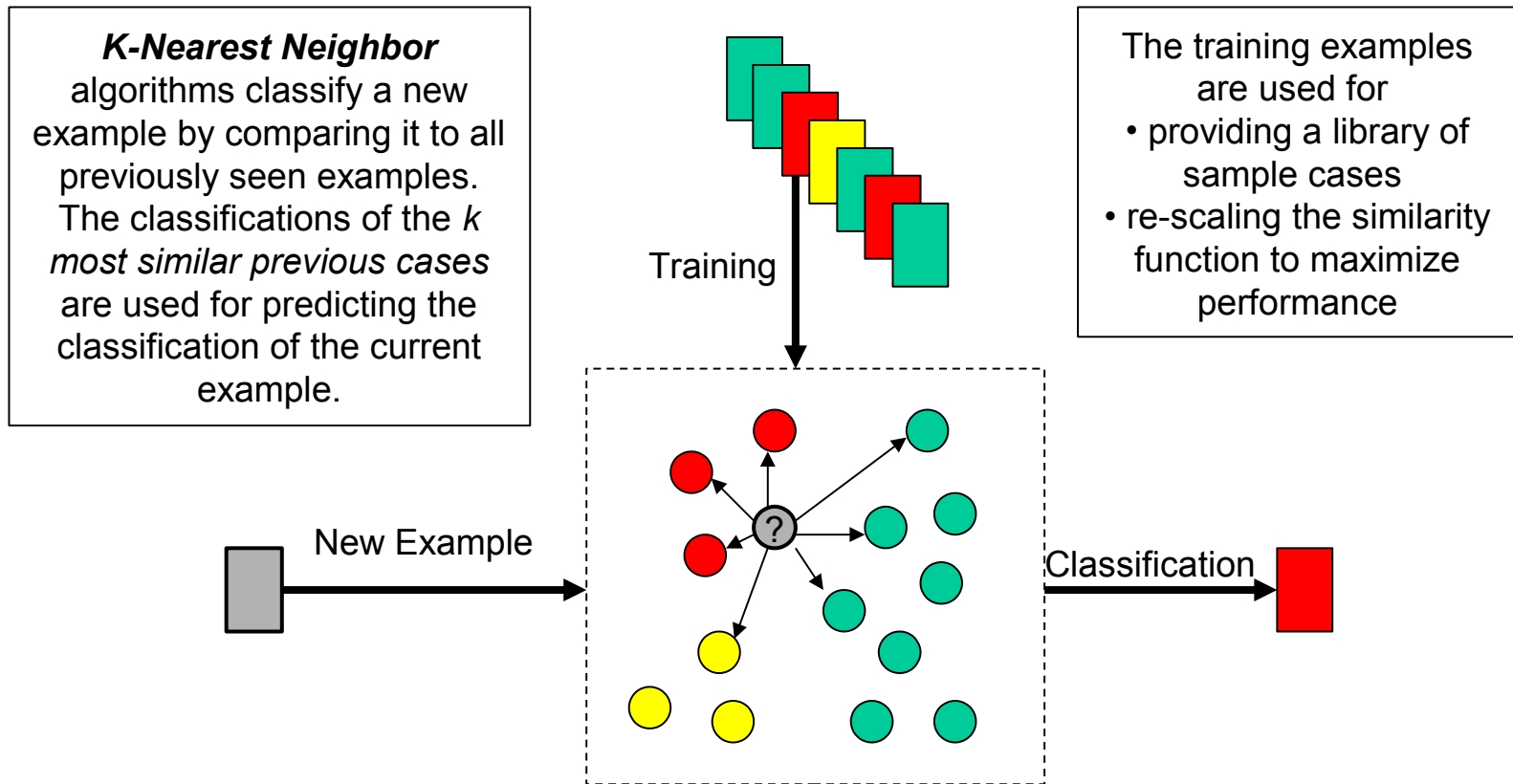
Nearest Neighbor Classification

Day	Temperature	Outlook	Humidity	Windy	Play Golf?
07-05	hot	sunny	high	false	no
07-06	hot	sunny	high	true	no
07-07	hot	overcast	high	false	yes
07-09	cool	rain	normal	false	yes
07-10	cool	overcast	normal	true	yes
07-12	mild	sunny	high	false	no
07-14	cool	sunny	normal	false	yes
07-15	mild	rain	normal	false	yes
07-20	mild	sunny	normal	true	yes
07-21	mild	overcast	high	true	yes
07-22	hot	overcast	normal	false	yes
07-23	mild	rain	high	true	no
07-26	cool	rain	normal	true	no
12-30	mild	rain	high	false	yes

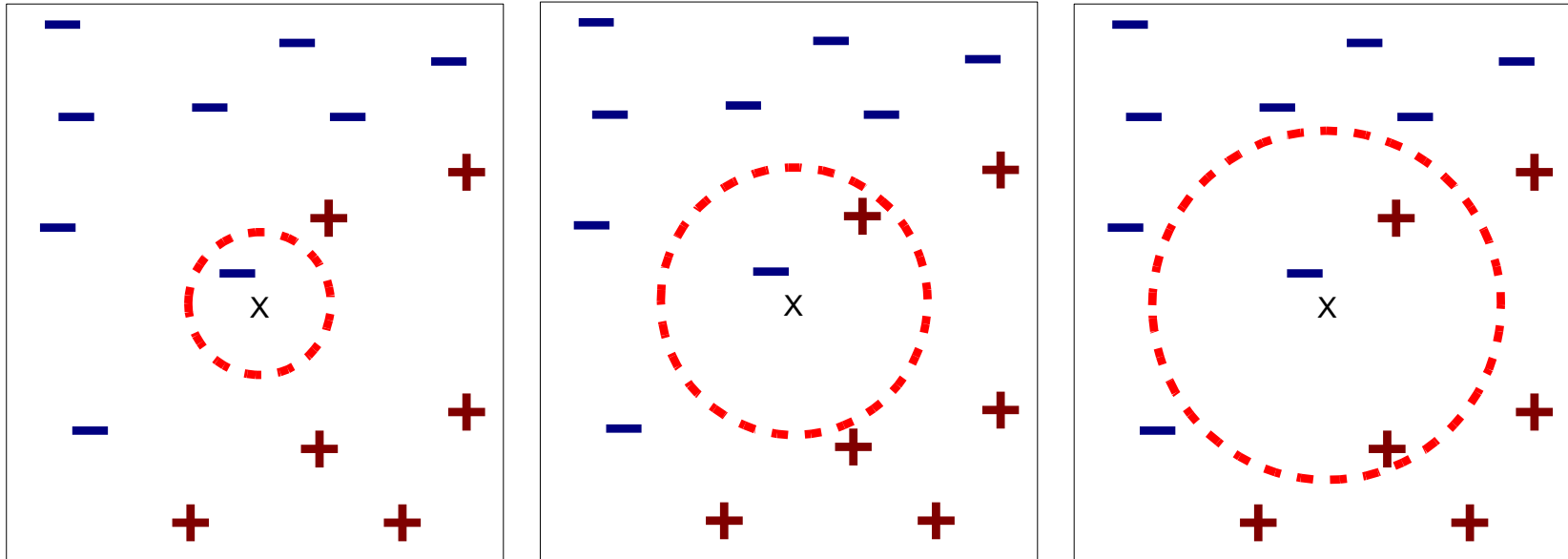
tomorrow	mild	sunny	normal	false	yes
----------	------	-------	--------	-------	-----



Nearest Neighbor Classifier



Nearest Neighbors



(a) 1-nearest neighbor

(b) 2-nearest neighbor

(c) 3-nearest neighbor

k nearest neighbors of an example x are the data points that have the k smallest distances to x



Prediction

The predicted class is determined from the nearest neighbor list

- **classification**

- take the majority vote of class labels among the k-nearest neighbors

$$\hat{y} = \max_c \sum_{i=1}^k \begin{cases} 1 & \text{if } y_i = c \\ 0 & \text{if } y_i \neq c \end{cases} = \max_c \sum_{i=1}^k \mathbf{1}(y_i = c)$$

indicator function

- can be easily be extended to **regression**

- predict the average value of the class value of the k-nearest neighbors

$$\hat{y} = \frac{1}{k} \sum_{i=1}^k y_i$$



Weighted Prediction

- Often prediction can be improved if the influence of each neighbor is weighted

$$\hat{y} = \frac{\sum_{i=1}^k w_i \cdot y_i}{\sum_{i=1}^k w_i}$$

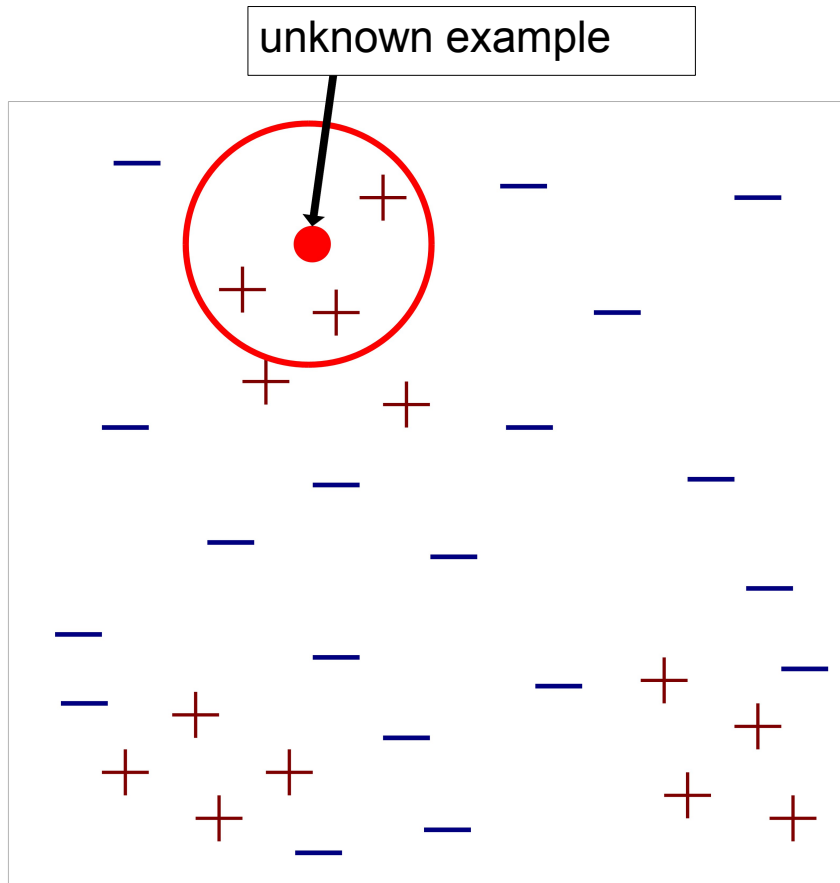
- Weights typically depend on distance, e.g.

$$w_i = \frac{1}{d(x_i, x)^2}$$

- Note:
 - with weighted distances, we could use all examples for classifications (→ **Inverse Distance Weighting**)



Nearest-Neighbor Classifiers

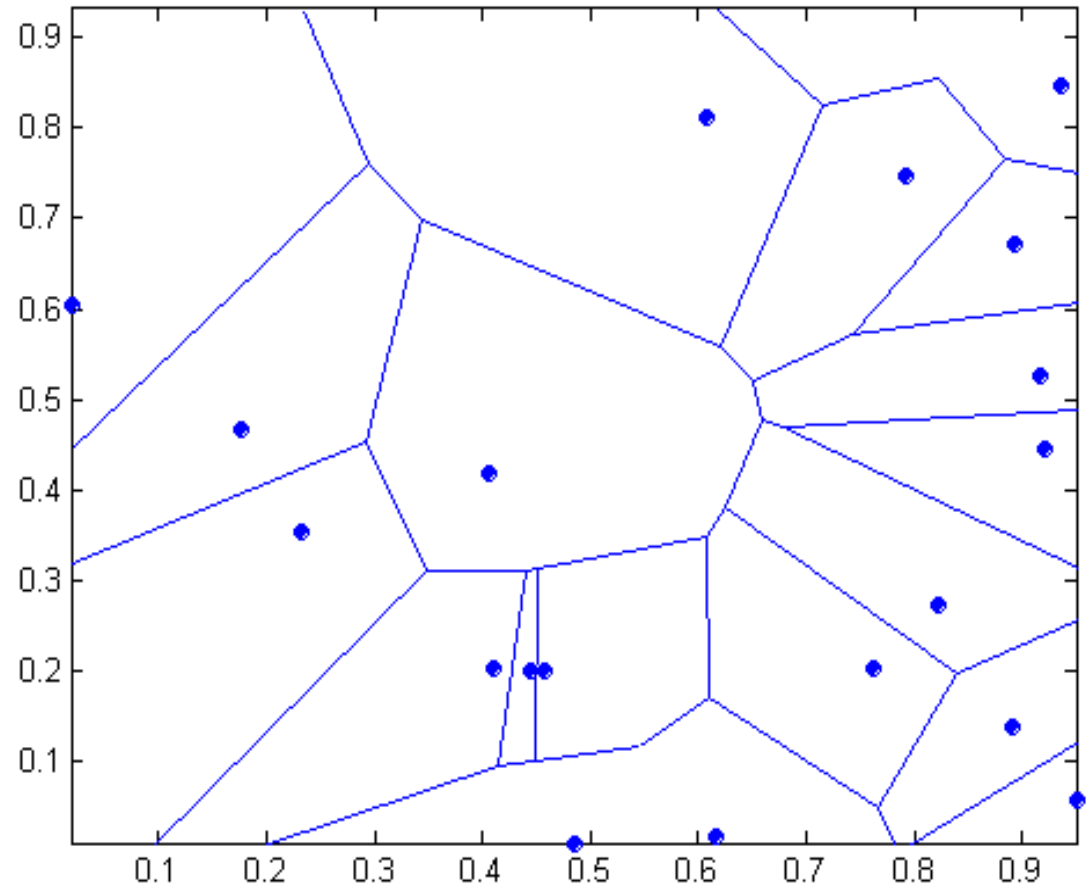


- Require three things
 - The set of stored examples
 - Distance Metric to compute distance between examples
 - The value of k , the number of nearest neighbors to retrieve
- To classify an unknown example:
 - Compute distance to other training examples
 - Identify k nearest neighbors
 - Use class labels of nearest neighbors to determine the class label of unknown example (e.g., by taking majority vote)



Voronoi Diagram

- shows the regions of points that are closest to a given set of points
- boundaries of these regions correspond to potential decision boundaries of 1NN classifier

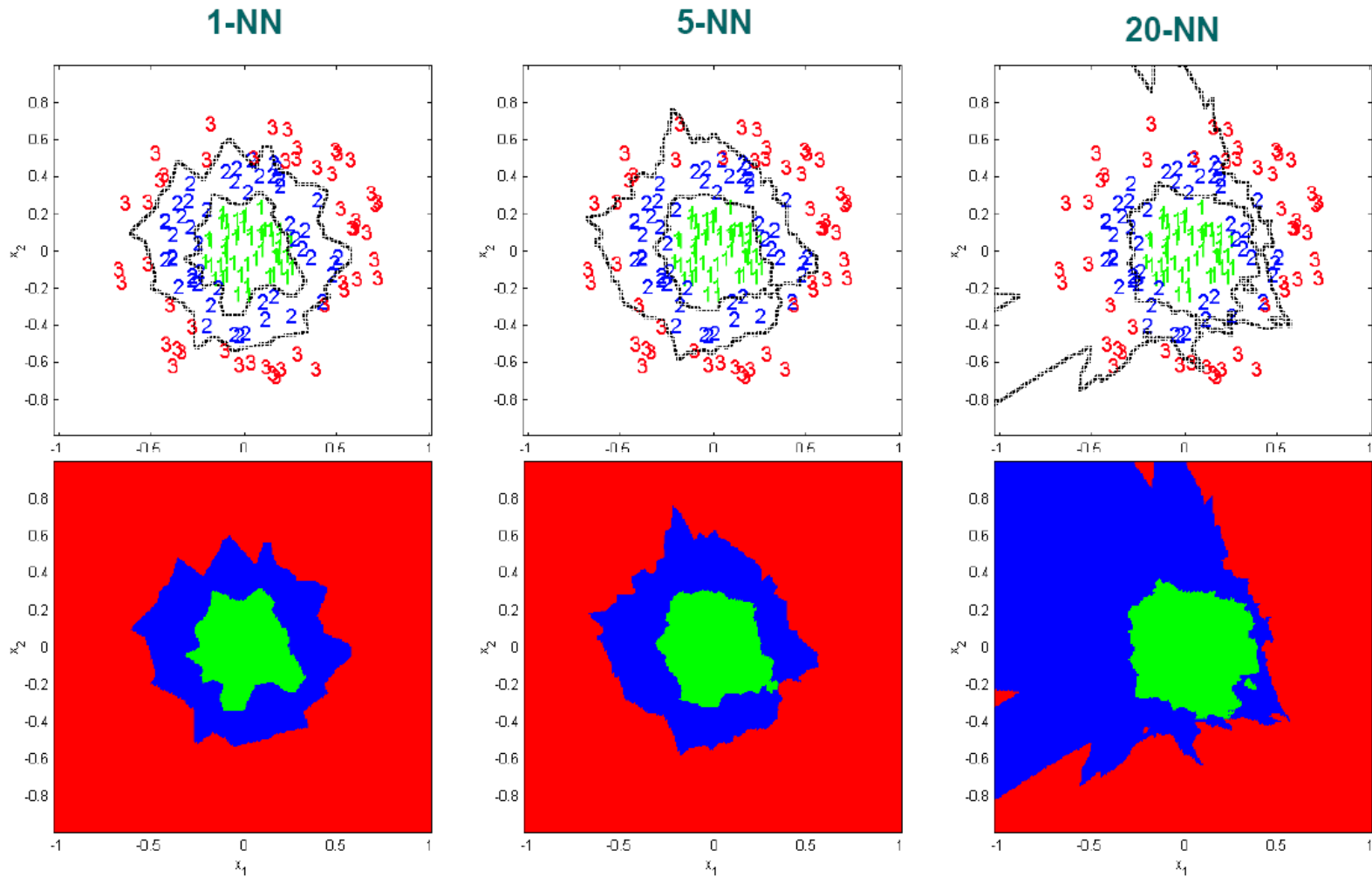


Lazy Learning Algorithms

- kNN is considered a **lazy learning** algorithm
 - Defers data processing until it receives a request to classify an unlabelled example
 - Replies to a request for information by combining its stored training data
 - Discards the constructed answer and any intermediate results
- Other names for lazy algorithms
 - Memory-based, Instance-based , Exemplar-based , Case-based, Experiencebased
- This strategy is opposed to **eager learning** algorithms which
 - Compiles its data into a compressed description or model
 - Discards the training data after compilation of the model
 - Classifies incoming patterns using the induced model

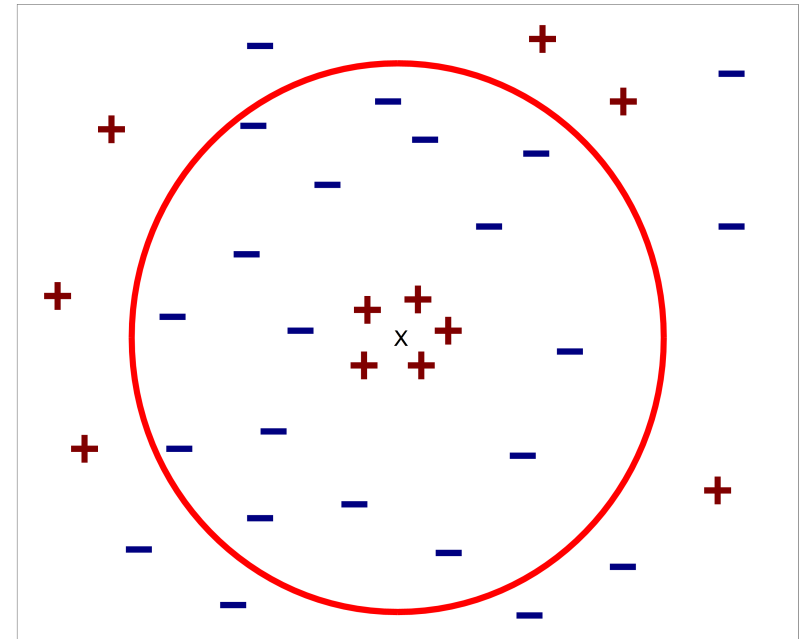


Choosing the value of k



Choosing the value of k

- If k is too small
 - sensitive to noise in the data (misclassified examples)
- If k is too large
 - neighborhood may include points from other classes
 - limiting case: $k \geq |D|$
 - all examples are considered
 - largest class is predicted
- good values can be found
 - e.g, by evaluating various values with cross-validation on the training data



Distance Functions for Numerical Attributes

- Numerical Attributes:
 - distance between two attribute values

$$d_A(v_1, v_2) = |v_1 - v_2|$$

- Normalization:
 - Different attributes are measured on different scales
→ values need to be normalized in $[0, 1]$:

$$\hat{v}_i = \frac{v_i - \min v_j}{\max v_j - \min v_j}$$

- Note:
 - This normalization assumes a (roughly) uniform distribution of attribute values
 - For other distributions, other normalizations might be preferable
 - e.g.: logarithmic for salaries?



Distance Functions for Symbolic Attributes

- 0/1 distance

$$d_A(v_1, v_2) = \begin{cases} 0 & \text{if } v_1 = v_2 \\ 1 & \text{if } v_1 \neq v_2 \end{cases}$$

- Value Difference Metric (VDM) (Stanfill & Waltz 1986)

- two values are similar if they have approximately the same distribution over all classes (similar relative frequencies in all classes)
- sum over all classes the difference of the percentage of examples with value v_1 in this class and examples with value v_2 in this class

$$d_A(v_1, v_2) = \sum_c \left| \frac{n_{1,c}}{n_1} - \frac{n_{2,c}}{n_2} \right|^k$$

k is a user-settable parameter (e.g., $k=2$)

- used in PEBLS with $k = 1$
(Parallel Exemplar-Based Learning System; Cost & Salzberg, 1993)



VDM Example

<i>Tid</i>	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Class	Refund	
	Yes	No
Yes	0	3
No	3	4

Distance between values:

$$\begin{aligned}
 d(\text{Refund}=\text{Yes}, \text{Refund}=\text{No}) \\
 &= |0/3 - 3/7| + |3/3 - 4/7| = 6/7
 \end{aligned}$$

VDM Example

<i>Tid</i>	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Class	Marital Status		
	Single	Married	Divorced
Yes	2	0	1
No	2	4	1

Distance between values:

$d(\text{Single}, \text{Married})$

$$= |2/4 - 0/4| + |2/4 - 4/4| = 1$$

$d(\text{Single}, \text{Divorced})$

$$= |2/4 - 1/2| + |2/4 - 1/2| = 0$$

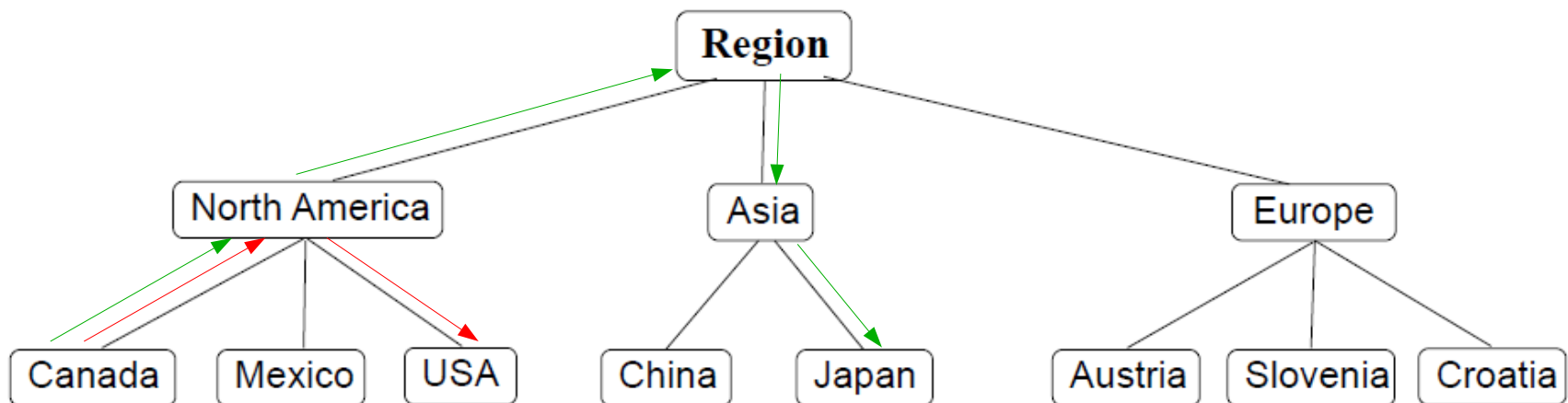
$d(\text{Married}, \text{Divorced})$

$$= |0/4 - 1/2| + |4/4 - 1/2| = 1$$



Other Distance Functions

- Other distances are possible
 - hierarchical attributes
 - distance of the values in the hierarchy
 - e.g., length of shortest path from v_1 to v_2



$$d(\text{Canada}, \text{USA})=2, d(\text{Canada}, \text{Japan})=4$$



Other Distance Functions

- Other distances are possible
 - hierarchical attributes
 - distance of the values in the hierarchy
 - e.g., length of shortest path from v_1 to v_2
 - string values
 - edit distance

Virginia
Verginia
Verminia
Vermonia
Vermonta
Vermont

$$d(\text{Virginia}, \text{Vermont}) = 5$$



Other Distance Functions

- Other distances are possible
 - hierarchical attributes
 - distance of the values in the hierarchy
 - e.g., length of shortest path from v_1 to v_2
 - string values
 - edit distance
- in general
 - distances are domain-dependent
 - can be chosen appropriately

Distances for Missing Values

- not all attribute values may be specified for an example
- Common policy:
 - assume missing values to be maximally distant



Feature Weighting

- Not all dimensions are equally important
 - comparisons on some dimensions might even be completely irrelevant for the prediction task
 - straight-forward distance functions give equal weight to all dimensions
- Idea:
 - use a weight for each attribute to denote its importance
 - e.g., **Weighted** Euclidean Distance:

$$d(x_1, x_2) = \sqrt{\sum_A w_A \cdot d_A(v_{1,A}, v_{2,A})^2}$$

- weights w_A can be set by user or determined automatically
- Survey of feature weighting algorithms:

Dietrich Wettschereck, David W. Aha, Takao Mohri:

[A Review and Empirical Evaluation of Feature Weighting Methods for a Class of Lazy Learning Algorithms.](#)

Artificial Intelligence Review 11(1-5): 273-314 (1997)



RELIEF

(Kira & Rendell, ICML-92)

Basic idea:

in a local neighborhood around an example x a good attribute A should

- allow to discriminate x from all examples of different classes (the set of *misses*)
 - therefore the probability that the attribute has a different value for x and a miss m should be high
- have the same value for all examples of the same class as x (the set of *hits*)
 - therefore the probability that the attribute has a different value for x and a hit h should be low

→ try to estimate and maximize

$$w_A = Pr(v_x \neq v_m) - Pr(v_x \neq v_h)$$

where v_x is the value of attribute A in example x

- this probability can be estimated via the average distance



RELIEF

(Kira & Rendell, ICML-92)

1. set all attribute weights $w_A = 0.0$
2. for $i = 1$ to r (\leftarrow user-settable parameter)
 - select a random example x
 - find
 - h : nearest neighbor of same class (*near hit*)
 - m : nearest neighbor of different class (*near miss*)
 - for each attribute A

$$w_A \leftarrow w_A + \frac{1}{r} \cdot (d_A(m, x) - d_A(h, x))$$

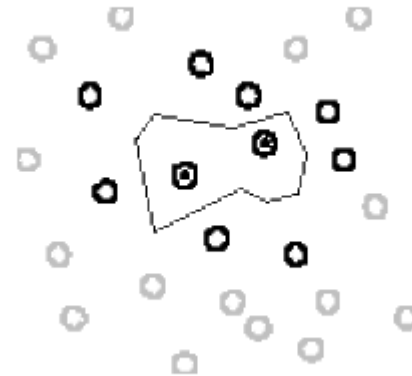
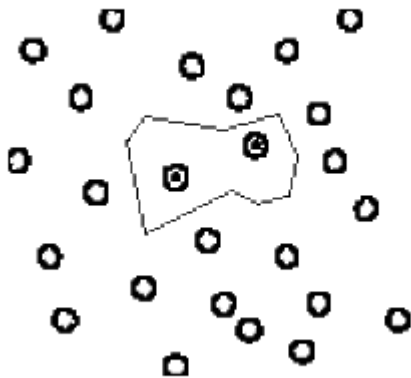
where $d_A(x, y)$ is the distance in attribute A between examples x and y (normalized to $[0, 1]$ -range).

Note: when used for feature weighting, all $w_A < 0.0$ are set to 0 in the end.



Learning Prototypes

- Only those instances involved in a decision need to be stored
 - Noisy instances should be filtered out
- Idea:
 - only use prototypical examples



Learning Prototypes: IB-algorithms

- Case Study for prototype selection
 - Aha, Kibler and Albert: Instance-based learning. *Machine Learning*, 1991.
- **IB1:** Store all examples
 - high noise tolerance
 - high memory demands
- **IB2:** Store new example only if misclassified by stored examples
 - low noise tolerance
 - low memory demands
- **IB3:** like IB2, but
 - maintain a counter for the number of times the example participated in correct and incorrect classifications
 - use a significant test for filtering noisy examples
 - improved noise tolerance
 - low memory demands



- Idea:
 - we assign a weight to each instance
 - instances with lower weights are always distant
 - hence have a low impact on classification
 - instance weight $w_x=0$ completely ignores this instance x

→ Selecting instances is a special case of instance weighting
- Similarity function used in PEBLS (Cost & Salzberg, 1993)

$$d(x_1, x_2) = \frac{1}{w_{x_1} \cdot w_{x_2}} \cdot \sum_A d_A(v_1, v_2)^k$$

where $w_x = \frac{\text{Number of times } x \text{ has correctly predicted the class}}{\text{Number of times } x \text{ has been used for prediction}}$

- $w_x \approx 1$ if instance x predicts well
- $w_x < 1$ if instance x does not predict well



Efficiency of NN algorithms

- very efficient in training
 - only store the training data
- not so efficient in testing
 - computation of distance measure to every training example
 - much more expensive than, e.g., rule learning
- Note that kNN and 1NN are equal in terms of efficiency
 - retrieving the k nearest neighbors is (almost) no more expensive than retrieving a single nearest neighbor
 - k nearest neighbors can be maintained in a queue

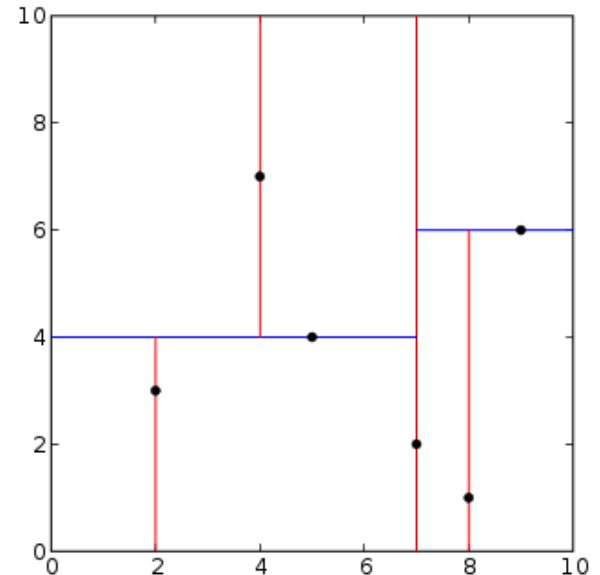
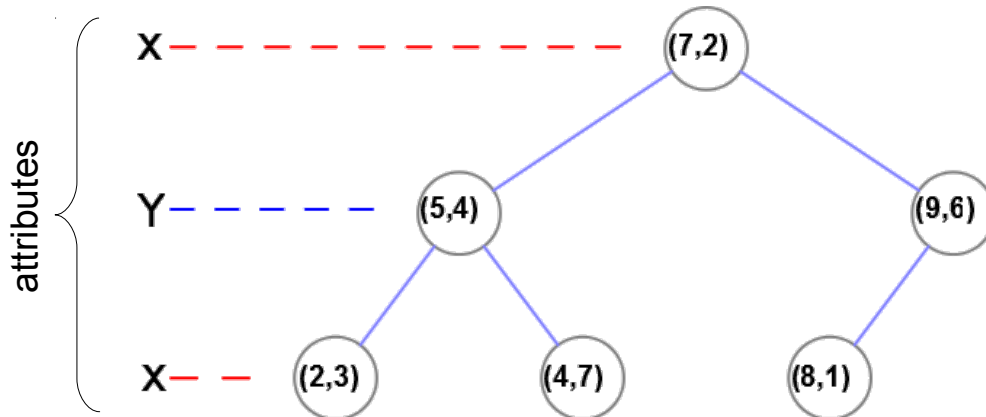


Finding nearest neighbors efficiently

- Simplest way of finding nearest neighbour:
 - linear scan of the data
 - classification takes time proportional to the product of the number of instances in training and test sets
- Nearest-neighbor search can be done more efficiently using appropriate data structures
 - kD-trees
 - ball trees



- common setting (others possible)
 - each level corresponds to one of the attributes
 - order of attributes can be arbitrary, fixed, and cyclic
 - each level splits according to this attribute
 - ideally use the median value (results in balanced trees)
 - often simply use the value of the next example



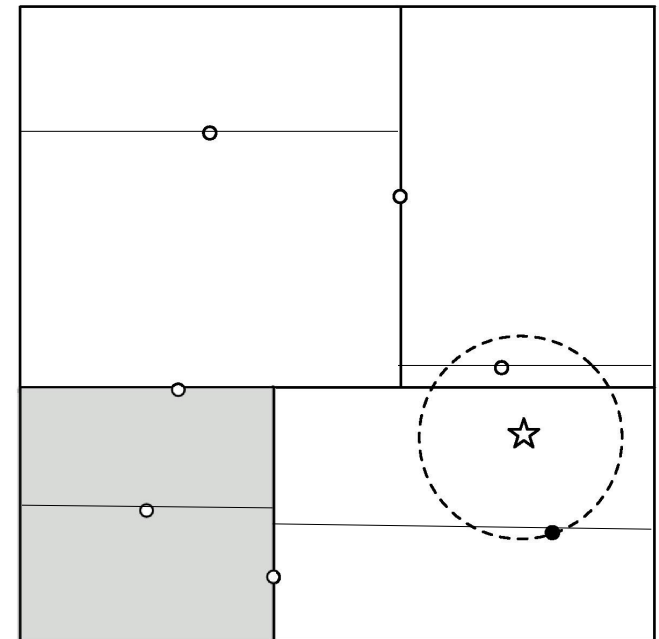
Building kD-trees incrementally

- Big advantage of instance-based learning: classifier can be updated incrementally
 - Just add new training instance after it arrives!
- Can we do the same with kD-trees?
- Heuristic strategy:
 - Find leaf node containing new instance
 - If leaf is empty
 - place instance into leaf
 - Else
 - split leaf according to the next dimension
 - Alternatively: split according to the longest dimension
 - idea: preserve squareness
- Tree should be re-built occasionally
 - e.g., if depth grows to twice the optimum depth



Using kD-trees: example

- The effect of a kD-tree is to **partition** the (multi-dimensional) sample space **according to** the underlying **data distribution**
 - finer partitioning in regions with high density
 - coarser partitioning in regions with low density
 - For a given **query** point
 - descending the tree to **find the** data points lying in the **cell** that contains the query point
 - **examine surrounding cells** if they overlap the ball centered at the query point and the closest data point so far
 - recursively back up one level and check distance to the split point
 - if overlap also search other branch
- only a few cells have to be searched



Using kD-trees: example

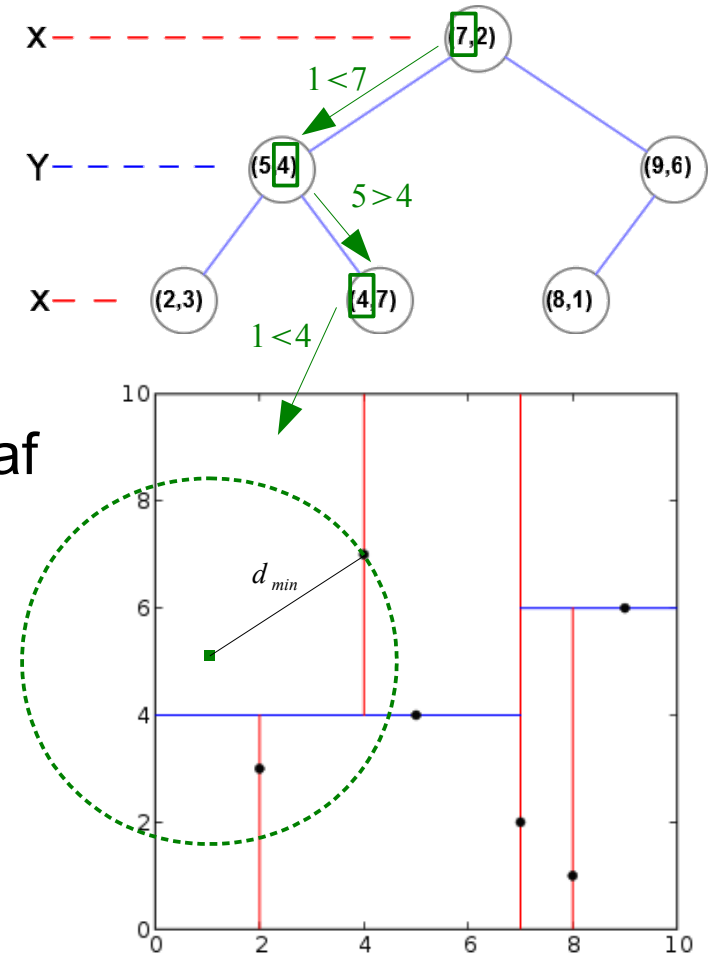
- Assume we have example [1,5]
 - Unweighted Euclidian distance

$$d(e_1, e_2) = \sqrt{\sum_A d_A(e_1, e_2)^2}$$

- sort the example down the tree:
 - ends in the left successor of [4,7]
- compute distance to example in the leaf

$$d([1,5], [4,7]) = \sqrt{(1-4)^2 + (5-7)^2} = \sqrt{13}$$

- now we have to look into rectangles that may contain a nearer example
 - remember the difference to the closest example $d_{min} = \sqrt{13}$



Using kD-trees: example

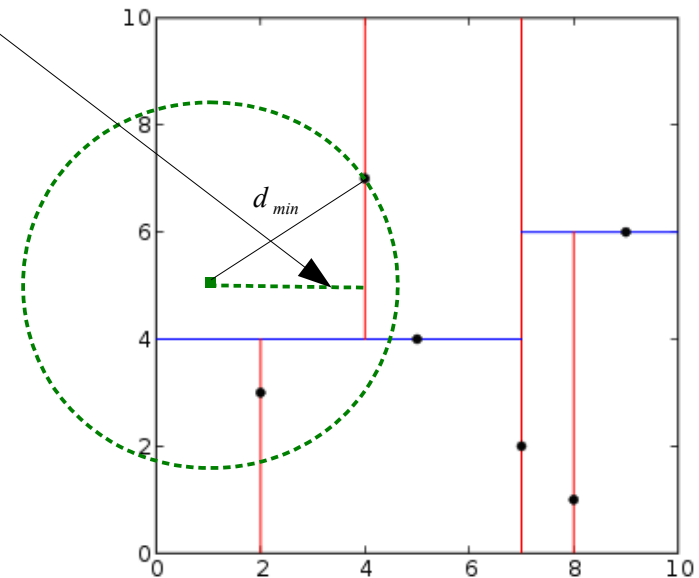
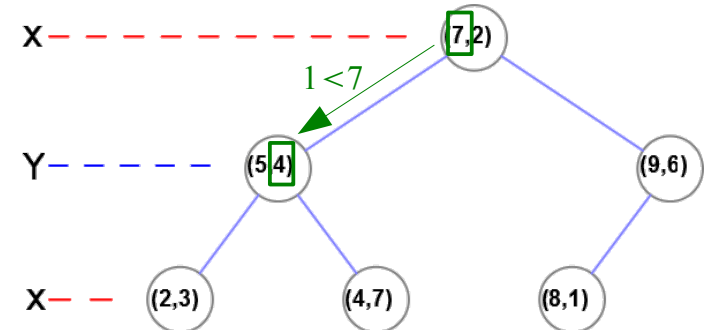
- go up one level (to example [4,7])
- compute distance to the closest point on this split (difference only on X)

$$d([1,5], [4, *]) = \sqrt{(4-1)^2 + 0^2} = 3$$

- If the difference is smaller than the current best difference

$$d([1,5], [4, *]) = 3 < \sqrt{13} = d_{min}$$

- then we could have a closer example in the right subtree of [4,7]
 - which in our case does not contain any example → done



Using kD-trees: example

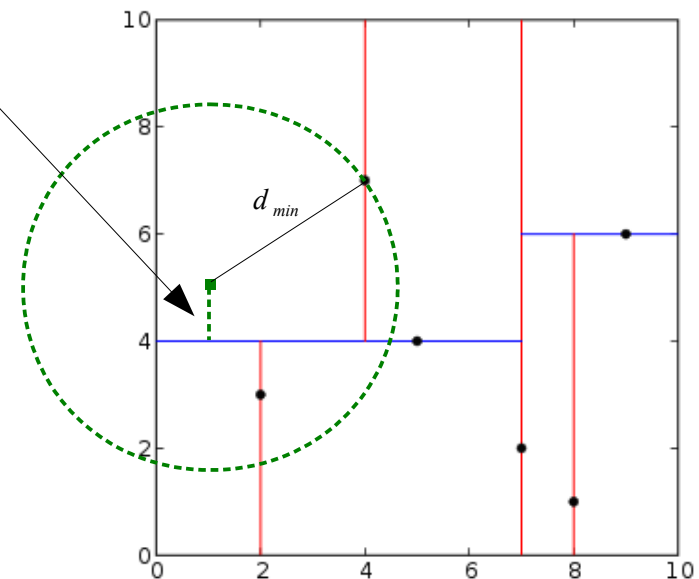
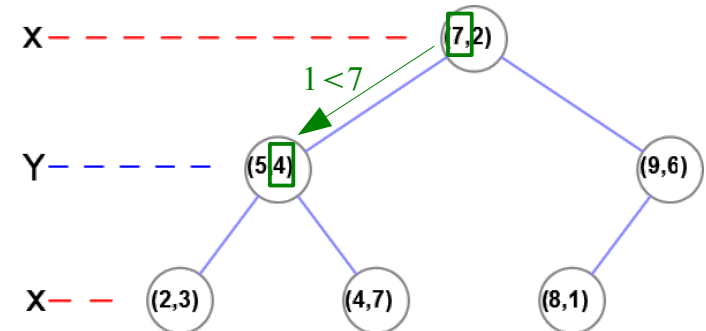
- go up one level (to example [5,4])
- compute distance to the closest point on this split (difference only on Y)

$$d([1,5], [* ,4]) = \sqrt{0^2 + (5-4)^2} = 1$$

- if the difference is smaller than the current best difference

$$d([1,5], [* ,4]) = 1 < \sqrt{13} = d_{min}$$

- then we could have a closer example in area $Y < 4$.
 - go down the other branch
 - and repeat recursively



Using kD-trees: example

- go down to leaf [2,3]
- compute distance to example in this leaf

$$d([1,5],[2,3]) = \sqrt{(1-2)^2 + (5-3)^2} = \sqrt{5}$$

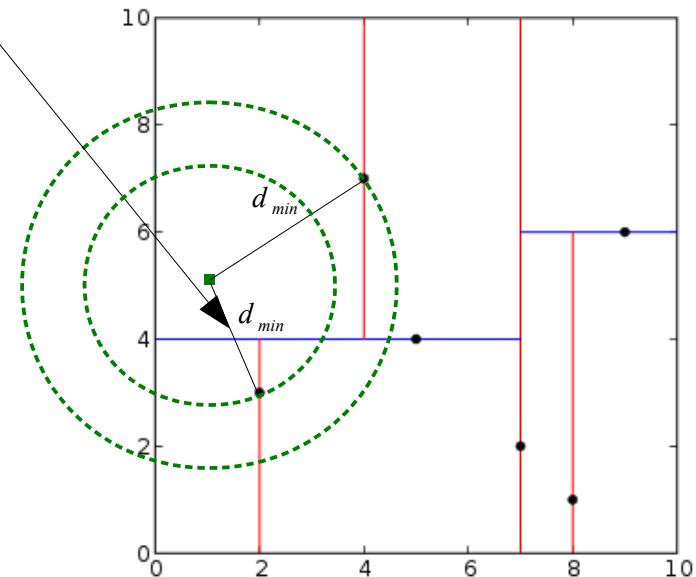
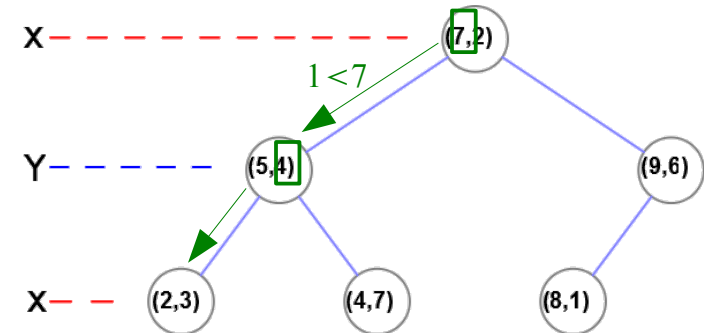
- if the difference is smaller than the current best difference

$$d([1,5],[2,3]) = \sqrt{5} < \sqrt{13} = d_{min}$$

- then the example in the leaf is the new nearest neighbor and

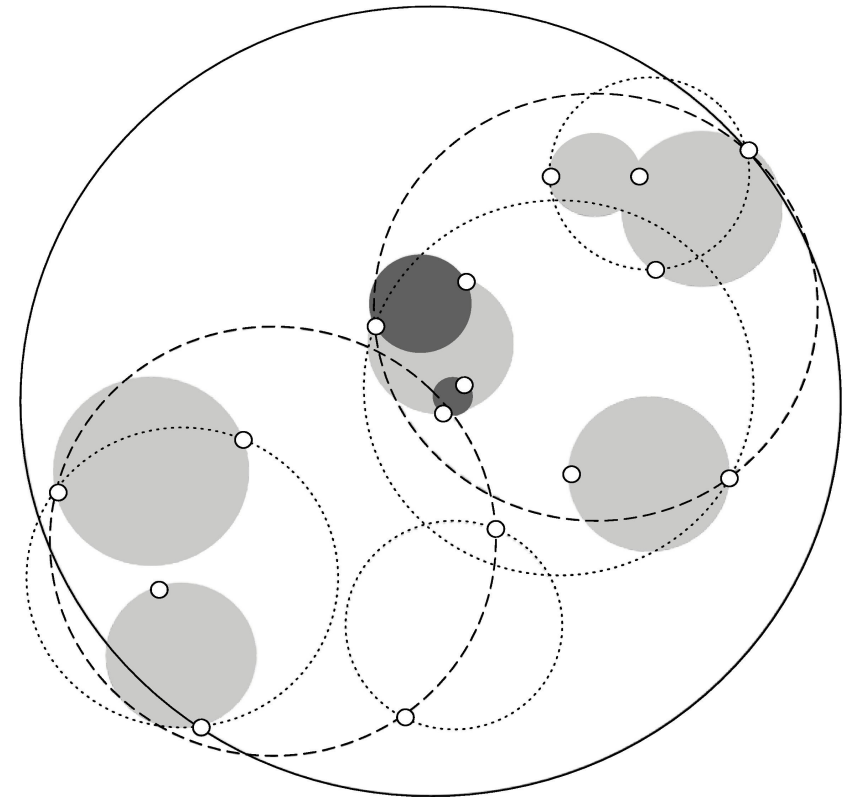
$$d_{min} = \sqrt{5} < \sqrt{13}$$

- this is recursively repeated until we have processed the root node
 - no more distances have to be computed



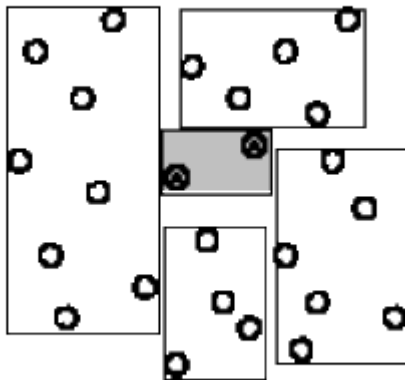
Ball trees

- Problem in kD-trees: corners
 - Observation:
 - There is no need to make sure that regions don't overlap
- We can use balls (hyperspheres) instead of hyperrectangles
- A ball tree organizes the data into a tree of k-dimensional hyperspheres
 - Normally allows for a better fit to the data and thus more efficient search

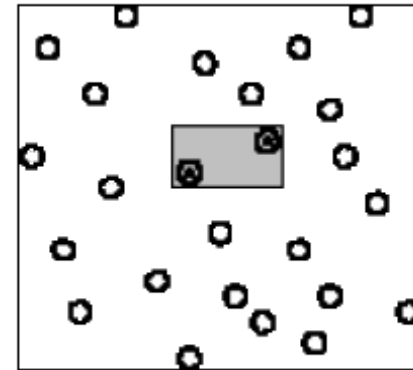


Nearest Hyper-Rectangle

- Nearest-Neighbor approaches can be extended to compute the distance to the nearest hyper-rectangle
 - a hyper-rectangle corresponds to a rule
 - conditions are intervals along each dimension



non-overlapping rectangles

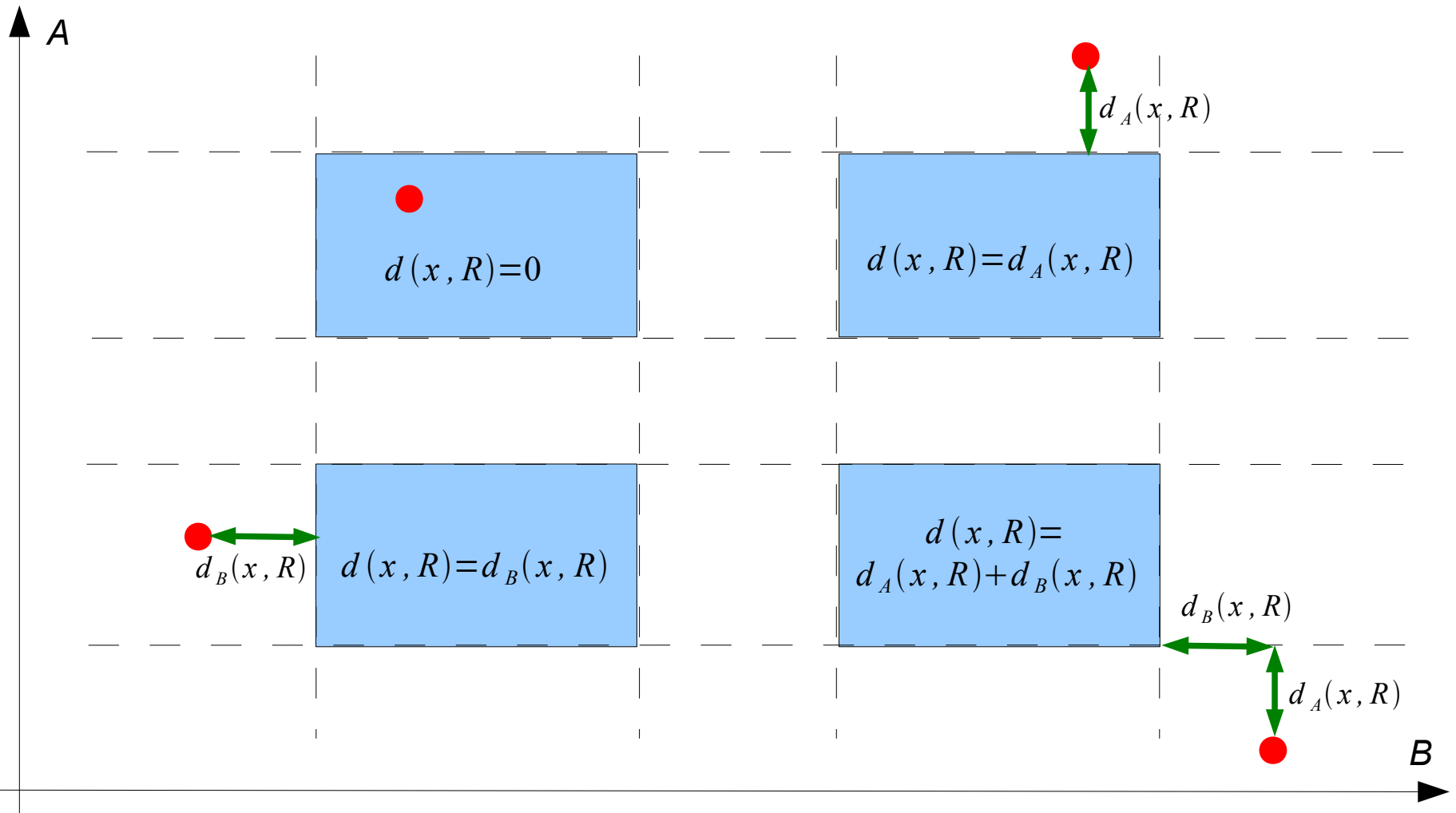


nested rectangles

- To do so, we need to adapt the distance measure
 - distance of a point to a rectangle instead of point-to-point distance



Rectangle-to-Point Distance



Rectangle-to-Point Attribute Distance

- numeric Attributes

- distance of the point to the closest edge of the rectangle along this attribute (i.e., distance to the upper/lower bound of the interval)

$$d_A(v, R) = \begin{cases} 0 & \text{if } v_{min, A_R} \leq v \leq v_{max, A_R} \\ v - v_{max, A_R} & \text{if } v > v_{max, A_R} \\ v_{min, A_R} - v & \text{if } v < v_{min, A_R} \end{cases}$$

if rule R uses $v_{min, A_R} \leq A \leq v_{max, A_R}$ as condition for attribute A

- symbolic attributes

- 0/1 distance $d_A(v, R) = \begin{cases} 0 & \text{if } v = v_{A_R} \\ 1 & \text{if } v \neq v_{A_R} \end{cases}$

if rule R uses $A = v_{A_R}$ as condition for attribute A

One can also adapt other distances. RISE uses a version of the VDM.



NEAR (Salzberg, 1991)

1. randomly choose r seed examples
 - convert them into rules
2. for each example x
 - choose rule $R_{min} = \arg \min_R d(x, R)$
 - if x is classified correctly by R_{min}
 - enlarge the condition of R_{min} so that x is covered
 - for each numeric attribute enlarge the interval if necessary
 - for each symbolic attribute delete the condition if necessary
 - else if x is classified incorrectly by R_{min}
 - add example x as a new rule

NEAR uses both instance and feature weighting

$$d(x, R) = w_x \sqrt{\sum_A w_A^2 d_A(x, R)^2}$$



Distance Functions

- Computes the distance between two examples
 - so that we can find the “nearest neighbor” to a given example
- General Idea:
 - reduce the distance $d(x_1, x_2)$ of two examples to the distances $d_A(v_1, v_2)$ between two values for attribute A

- Popular choices

- **Euclidean Distance:** $d(x_1, x_2) = \sqrt{\sum_A d_A(v_{1,A}, v_{2,A})^2}$
 - straight-line between two points

- **Manhattan or City-block Distance:** $d(x_1, x_2) = \sum_A d_A(v_{1,A}, v_{2,A})$
 - sum of axis-parallel line segments



Instance and Feature Weighting in NEAR

Instance Weighting as in PEBLS

Feature Weights are computed incrementally

- if an example is **incorrectly** classified
 - the weights of all matching attributes are increased by a fixed percentage (20%)
 - this has the effect of moving the example farther away along these dimensions
 - the weights of all attributes that do not match are decreased by a fixed percentage (20%)
- if an example is **correctly** classified
 - do the opposite (decrease matching and increase non-matching weights analogously)



Second Chance Heuristic

An improved version used a **Second Chance Heuristic**

- if the nearest rule did not classify correctly, try the second one
 - if this one matches → expand it to cover the example
 - if not → add the example as a new rule
- this can lead to the generation of nested rules
 - i.e., rectangles inside of other rectangles
 - at classification time, use the smallest matching rectangle
 - but this did not work well (overfitting?)
 - such nested rules may be interpreted as rules with exceptions



RISE (Domingos, 1996)

(Rule Induction from a Set of Exemplars)



1. turn each example into a rule resulting in a theory T
2. repeat
 - for each rule R in T
 - i. choose uncovered example $x_{min} = \arg \min_x d(x, R)$
 - ii. $R' = \text{minimalGeneralisation}(R, x_{min})$
 - iii. replace R with R' if this does not decrease the accuracy of T
 - iv. delete R' if it is already part of T (duplicate rule)
3. until no further increase in accuracy

- RISE uses the simple distance function

$$d(x, R) = \sum_A d_A(x, R)^k$$



RISE (Domingos, 1996)

- Classification of an example:
 - use the rule that is closest to the example
 - if multiple rules have the same distance, use the one with the highest Laplace-corrected precision
- Leave-one-out estimation of accuracy of a theory:
 - For classifying an example, the rule that encodes it is ignored
 - but only if it has not been generalized yet
 - can be computed efficiently if each examples remembers the distance to the rule by which it is classified
 - if a rule is changed, go once through all examples and see if the new rule classifies any examples that were classified by some other rule before
 - count the improvements (+1) or mistakes (-1) only for those examples, and see whether their sum is > 0 or < 0 .



Differences NEAR and RISE

■ NEAR

- focuses on examples
- incremental training
- instance weighted and feature-weighted Euclidean distance
- tie breaking using the smallest rule

■ RISE

- focuses on rules
- batch training
- straight-forward Manhattan distance
- tie breaking with Laplace heuristic



Discussion

- Nearest Neighbor methods are often very accurate
 - Assumes all attributes are equally important
 - Remedy: attribute selection or weights
 - Possible remedies against noisy instances
 - Take a majority vote over the k nearest neighbors
 - Removing noisy instances from dataset (difficult!)
 - Statisticians have used k-NN since early 1950s
 - If $n \rightarrow \infty$ and $k/n \rightarrow 0$, error approaches minimum
 - can model arbitrary decision boundaries
- ...but somewhat inefficient (at classification time)
 - straight-forward application maybe too slow
 - kD-trees become inefficient when number of attributes is too large (approximately > 10)
 - Ball trees work well in higher-dimensional spaces
- several similarities with rule learning

