# Instance-Based Learning

CS 760@UW-Madison

# Goals for the lecture

you should understand the following concepts

- $k$-NN classification
- $k$-NN regression
- edited nearest neighbor
- locally weighted regression
- inductive bias (hypothesis space bias, preference bias)

# Nearest-neighbor classification

learning stage

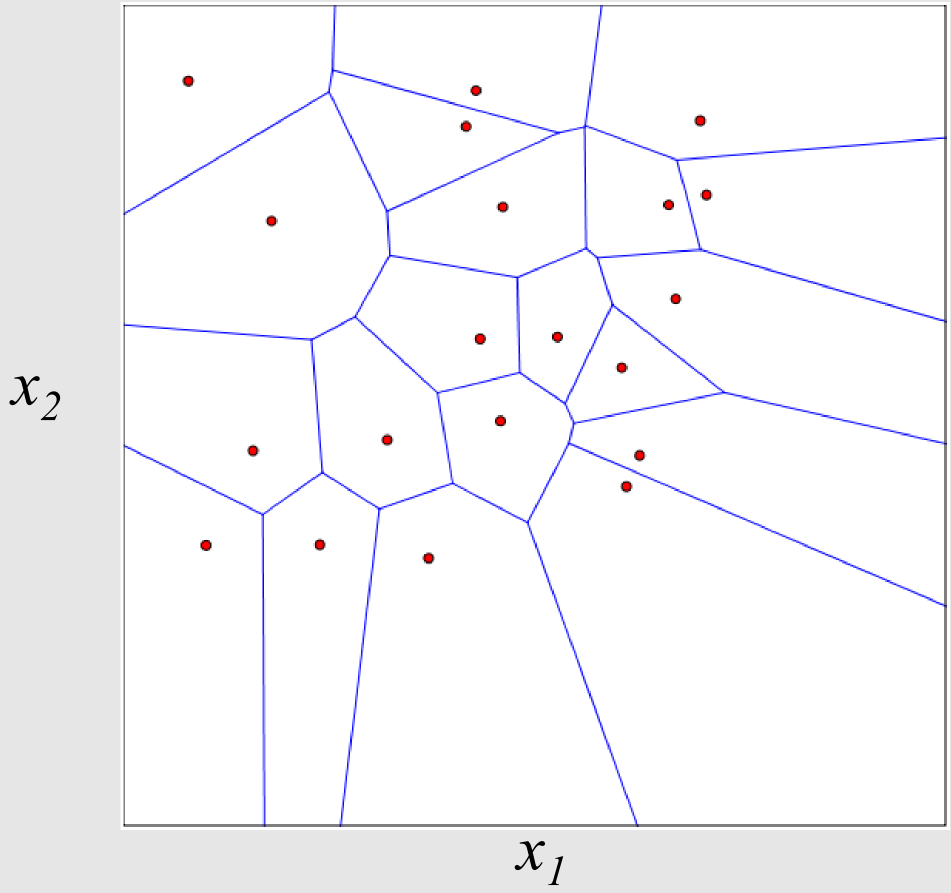
- given a training set $(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(m)}, y^{(m)})$, do nothing (it's sometimes called a *lazy learner*)

classification stage

- **given**: an instance $\boldsymbol{x}^{(q)}$ to classify
- find the training-set instance $\boldsymbol{x}^{(i)}$ that is most similar to $\boldsymbol{x}^{(q)}$
- return the class value $y^{(i)}$

# The decision regions

Voronoi diagram: each polyhedron indicates the region of feature space that is in the nearest neighborhood of each training instance



# $k$-nearest-neighbor classification

classification task

- **given**: an instance $\boldsymbol{x}^{(q)}$ to classify
- find the $k$ training-set instances $(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(k)}, y^{(k)})$ that are most similar to $\boldsymbol{x}^{(q)}$
- return the class value

$$\hat{y} \leftarrow \underset{v \in \text{values}(Y)}{\text{argmax}} \sum_{i=1}^{k} \delta(v, y^{(i)}) \qquad \delta(a,b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise} \end{cases}$$

(i.e. return the class that have the most instances)

# How can we determine distance

suppose all features are discrete

- Hamming distance: count the number of features for which two instances differ

suppose all features are continuous

- Euclidean distance:

$$d(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \sqrt{\sum_f \left(x_f^{(i)} - x_f^{(j)}\right)^2}$$ where $x_f^{(i)}$ represents the $f$-th feature of $x^{(i)}$

- Manhattan distance:

$$d(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \sum_f \left| x_f^{(i)} - x_f^{(j)} \right|$$

# How can we determine distance

- if we have a mix of discrete/continuous features:

$$d(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \sum_f \begin{cases} \left| x_f^{(i)} - x_f^{(j)} \right| & \text{if } f \text{ is continuous} \\ 1 - \delta\left(x_f^{(i)}, x_f^{(j)}\right) & \text{if } f \text{ is discrete} \end{cases}$$

- typically want to apply to continuous features some type of normalization (values range 0 to 1) or standardization (values distributed according to standard normal)

- many other possible distance functions we could use …

# Standardizing numeric features

- given the training set $D$, determine the mean and stddev for feature $x_i$

$$\mu_i = \frac{1}{|D|}\sum_{d=1}^{|D|} x_i^{(d)} \qquad \sigma_i = \sqrt{\frac{1}{|D|}\sum_{d=1}^{|D|}\left(x_i^{(d)} - \mu_i\right)^2}$$

- standardize each value of feature $x_i$ as follows

$$\hat{x}_i^{(d)} = \frac{x_i^{(d)} - \mu_i}{\sigma_i}$$

- do the same for test instances, using the same $\mu_i$ and $\sigma_i$ derived from the *training* data

# Variants

# $k$-nearest-neighbor *regression*

learning stage

- given a training set $(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(m)}, y^{(m)})$, do nothing

prediction stage

- **given**: an instance $\boldsymbol{x}^{(q)}$ to make a prediction for
- find the $k$ training-set instances $(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(k)}, y^{(k)})$ that are most similar to $\boldsymbol{x}^{(q)}$
- return the value

$$\hat{y} \leftarrow \frac{1}{k} \sum_{i=1}^{k} y^{(i)}$$

# Distance-weighted nearest neighbor

We can have instances contribute to a prediction according to their distance from $\boldsymbol{x}^{(q)}$

classification:

$$\hat{y} \leftarrow \underset{v \in \text{values}(Y)}{\operatorname{argmax}} \sum_{i=1}^{k} w_i \; \delta(v, y^{(i)})$$

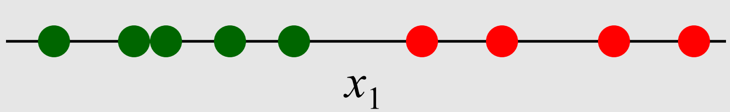
$$w_i = \frac{1}{d(x^{(q)}, x^{(i)})^2}$$

regression:

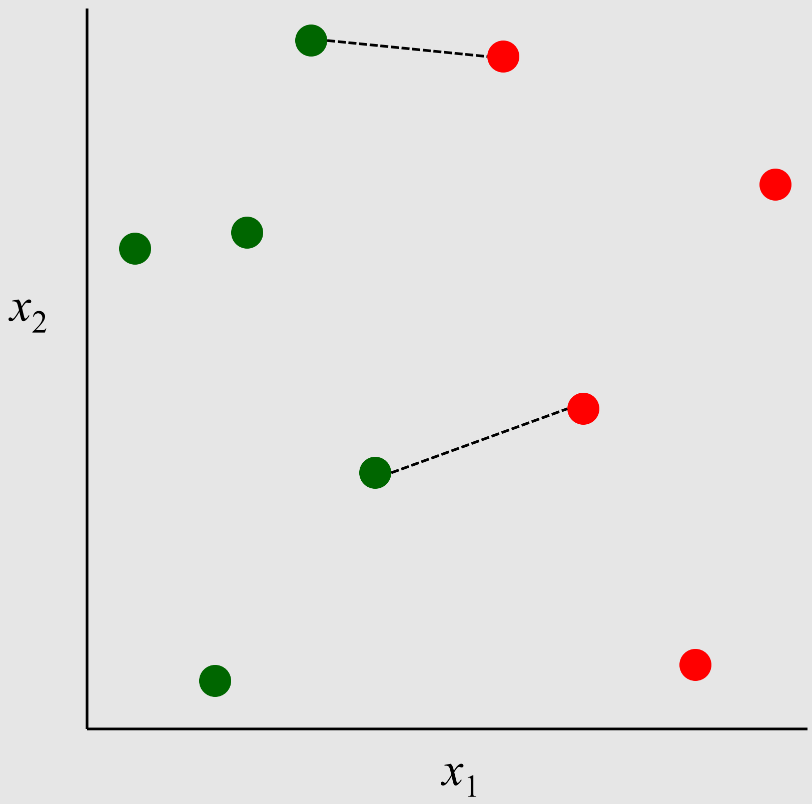
$$\hat{y} \leftarrow \frac{\sum_{i=1}^{k} w_i \; y^{(i)}}{\sum_{i=1}^{k} w_i}$$

# Irrelevant features

here's a case in which there is one relevant feature $x_1$ and a 1-NN rule classifies each instance correctly




consider the effect of an irrelevant feature $x_2$ on distances and nearest neighbors



# Locally weighted regression

- one way around this limitation is to weight features differently
- *locally weighted regression* is one nearest-neighbor variant that does this

prediction task

- **given**: an instance $\boldsymbol{x}^{(q)}$ to make a prediction for
- find the $k$ training-set instances $(\boldsymbol{x}^{(1)}, y^{(1)})$, …, $(\boldsymbol{x}^{(k)}, y^{(k)})$ that are most similar to $\boldsymbol{x}^{(q)}$
- return the value

$$f(\mathbf{x}^{(q)}) = w_0 + w_1 x_1^{(q)} + w_2 x_2^{(q)} + \ldots + w_n x_n^{(q)}$$

# Locally weighted regression

prediction/learning task

- find the weights $w_i$ for each $\boldsymbol{x}^{(q)}$ by minimizing

$$E(\mathbf{x}^{(q)}) = \sum_{i=1}^{k} \left( f(\mathbf{x}^{(i)}) - y^{(i)} \right)^2$$

- this is done at prediction time, specifically for $\boldsymbol{x}^{(q)}$
- can do this using gradient descent (to be covered soon)

# Speeding up $k$-NN

- $k$-NN is a “lazy” learning algorithm – does virtually nothing at training time
- but classification/prediction time can be costly when the training set is large

- two general strategies for alleviating this weakness
  - don’t retain every training instance (edited nearest neighbor)
  - use a smart data structure to look up nearest neighbors (e.g. a k-d tree)

# *Edited* instance-based learning

- select a subset of the instances that still provide accurate classifications

- *incremental deletion*

  start with all training instances in memory
  for each training instance $(\boldsymbol{x}^{(i)}, y^{(i)})$
      if other training instances provide correct classification for $(\boldsymbol{x}^{(i)}, y^{(i)})$
          delete it from the memory

- *incremental growth*

  start with an empty memory
  for each training instance $(\boldsymbol{x}^{(i)}, y^{(i)})$
      if other training instances in memory **don't** correctly classify $(\boldsymbol{x}^{(i)}, y^{(i)})$
          add it to the memory


Strength and Limitations

# Strengths of instance-based learning

- simple to implement
- “training” is very efficient
- adapts well to on-line learning
- robust to noisy training data (when $k > 1$)
- often works well in practice

# Limitations of instance-based learning

- sensitive to range of feature values
- sensitive to irrelevant and correlated features, although …
  - there are variants (such as locally weighted regression) that learn weights for different features
  - later we'll talk about *feature selection* methods
- classification/prediction can be inefficient, although edited methods and $k$-$d$ trees can help alleviate this weakness
- doesn't provide much insight into problem domain because there is no explicit model

# Inductive bias

- *inductive bias* is the set of assumptions a learner uses to be able to predict $y_i$ for a previously unseen instance $\boldsymbol{x}_i$
- two components
  - *hypothesis space bias*: determines the models that can be represented
  - *preference bias*: specifies a preference ordering within the space of models

- in order to *generalize* (i.e. make predictions for previously unseen instances) a learning algorithm must have an inductive bias

# Consider the inductive bias of DT and *k*-NN learners

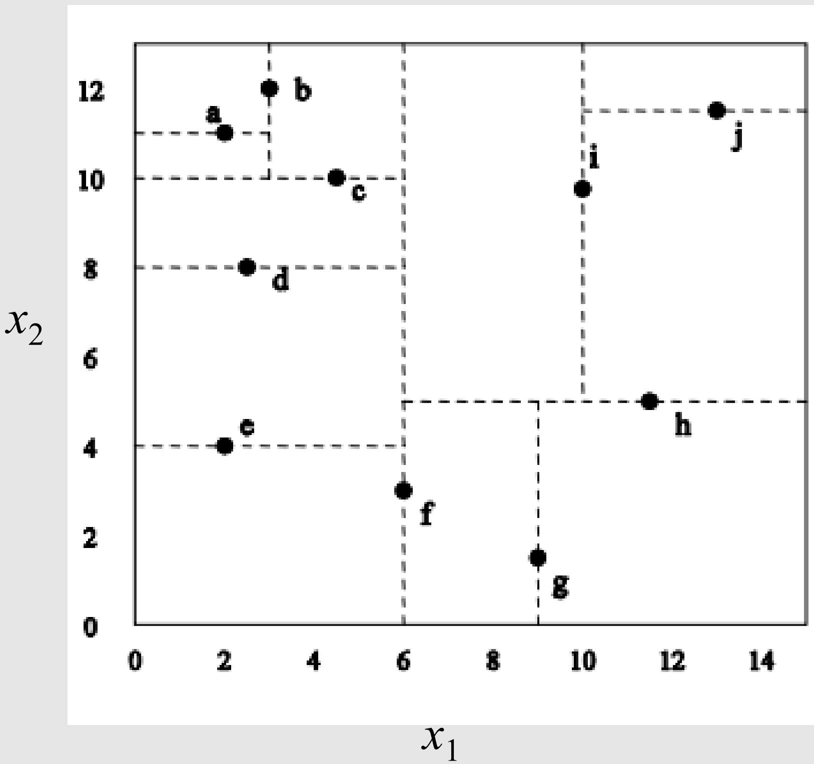
| learner | hypothesis space bias | preference bias |
|---|---|---|
| ID3 decision tree | trees with single-feature, axis-parallel splits | small trees identified by greedy search |
| *k*-NN | Voronoi decomposition determined by nearest neighbors | instances in neighborhood belong to same class |

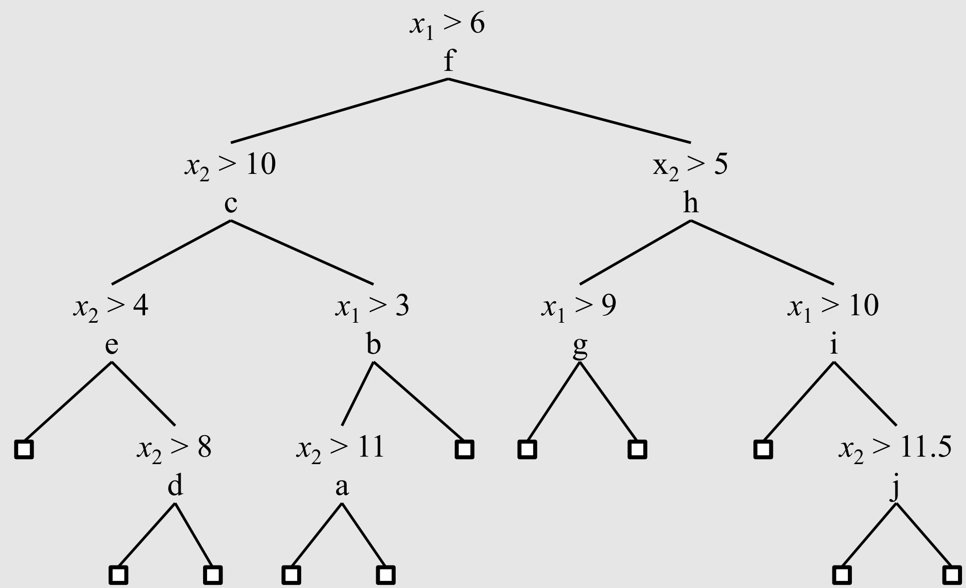
# Optional:
# k-d Tree: Data Structure for Finding Nearest Neighbors

# *k-d* trees

a *k-d tree* is similar to a decision tree except that each internal node

- stores one instance
- splits on the median value of the feature having the highest variance



# Finding nearest neighbors with a k-d tree

- use branch-and-bound search
- priority queue stores
  - nodes considered
  - lower bound on their distance to query instance

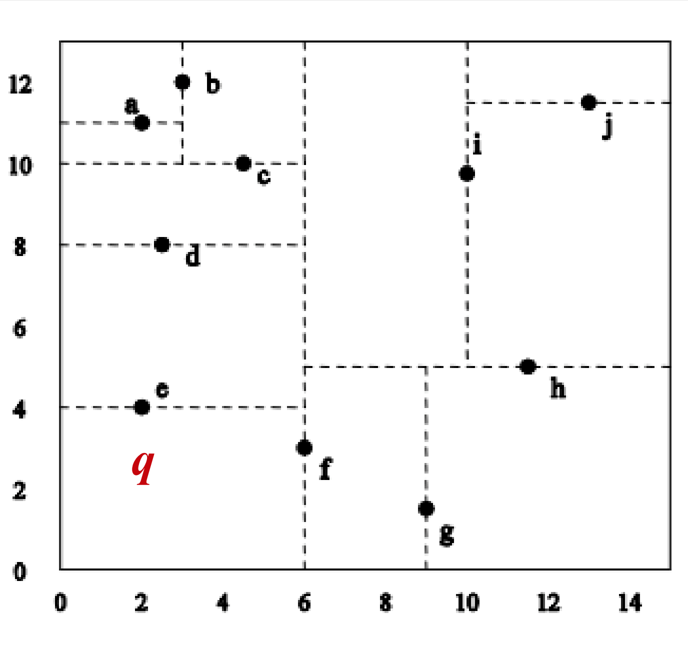
- lower bound given by distance using a single feature

- average case: $O(\log_2 m)$
- worst case: $O(m)$ where $m$ is the size of the training-set

# Finding nearest neighbors in a k-d tree

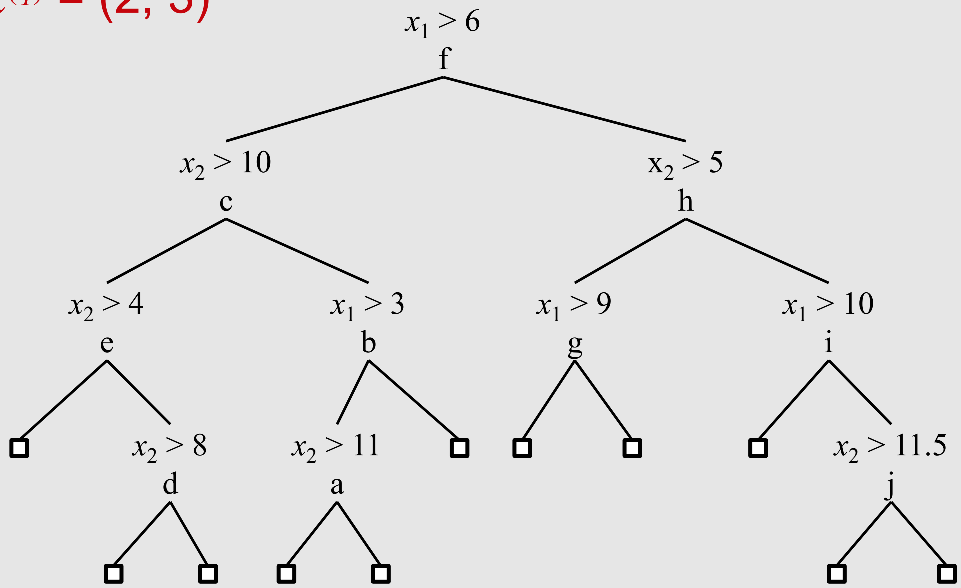
```
NearestNeighbor(instance x^(q))
  PQ = { }                                   // minimizing priority queue
  best_dist = ∞                              // smallest distance seen so far
  PQ.push(root, 0)
  while PQ is not empty
        (node, bound) = PQ.pop();
        if (bound ≥ best_dist)
                return best_node.instance    // nearest neighbor found
        dist = distance(x^(q), node. instance)
        if (dist < best_dist)
                best_dist = dist
                best_node = node
        if (q[node.feature] – node.threshold > 0)
                PQ.push(node.left, x^(q)[node.feature] – node.threshold)
                PQ.push(node.right, 0)
        else
                PQ.push(node.left, 0)
                PQ.push(node.right, node. threshold - x^(q) [node.feature])
  return best_node. instance
```

# k-d tree example (Manhattan distance)

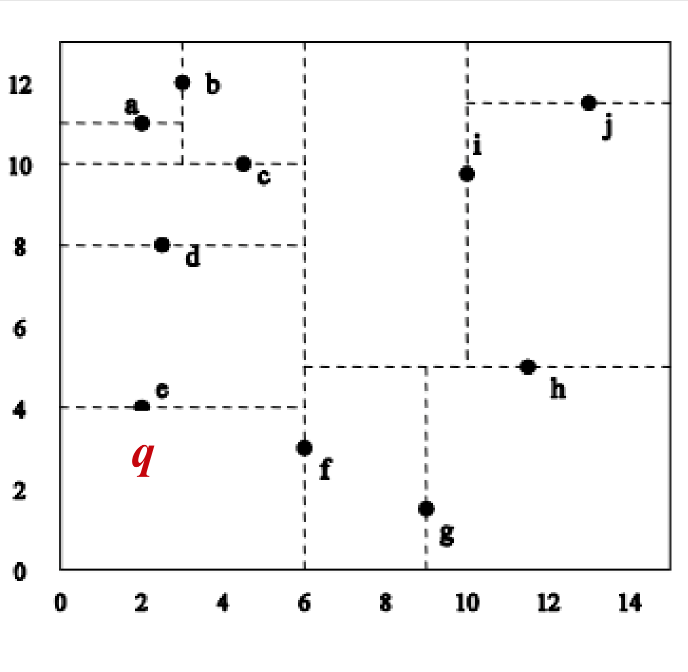



given query

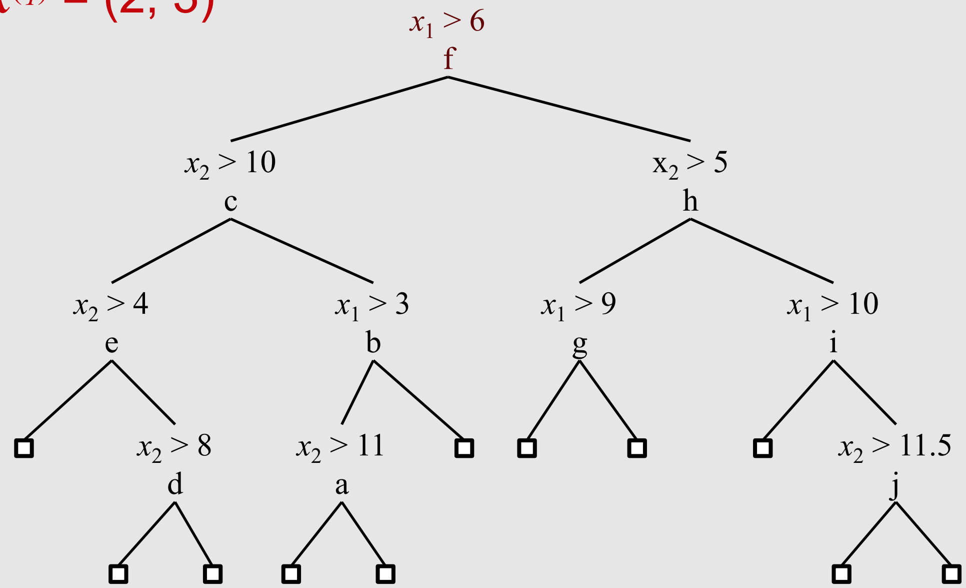
$\boldsymbol{x}^{(q)} = (2, 3)$



# k-d tree example (Manhattan distance)

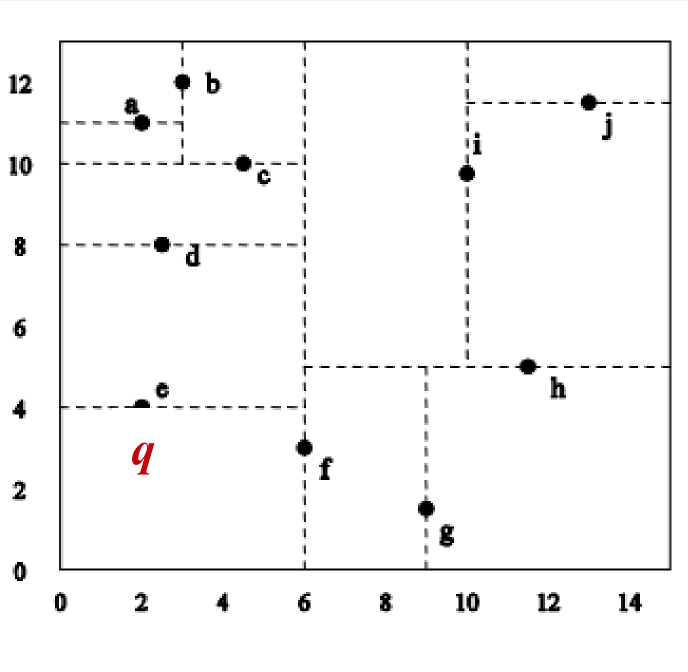



given query
$x^{(q)} = (2, 3)$

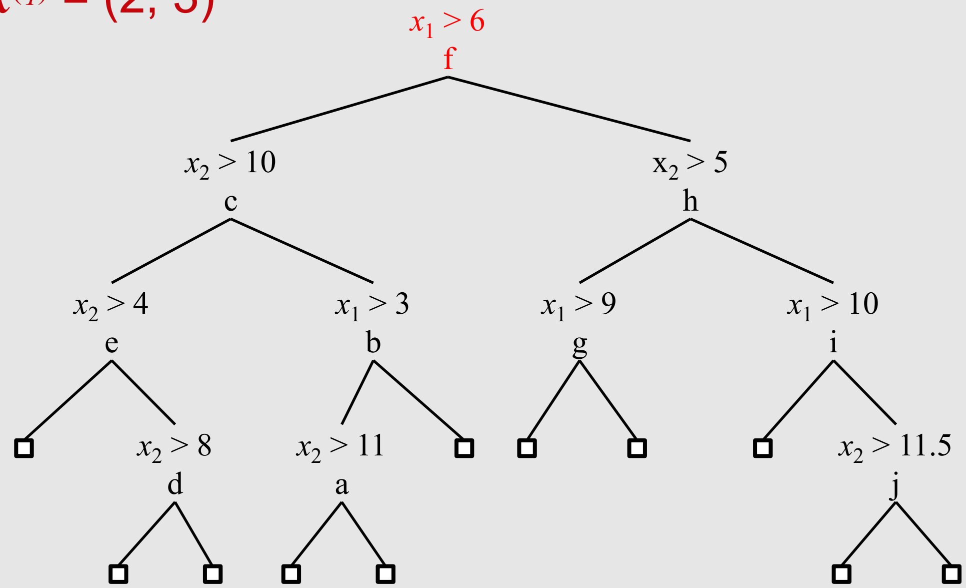



| distance | best distance | best node | priority queue |
|---|---|---|---|
| | ∞ | | (f, 0) |
| | | | |
| | | | |
| | | | |

# k-d tree example (Manhattan distance)



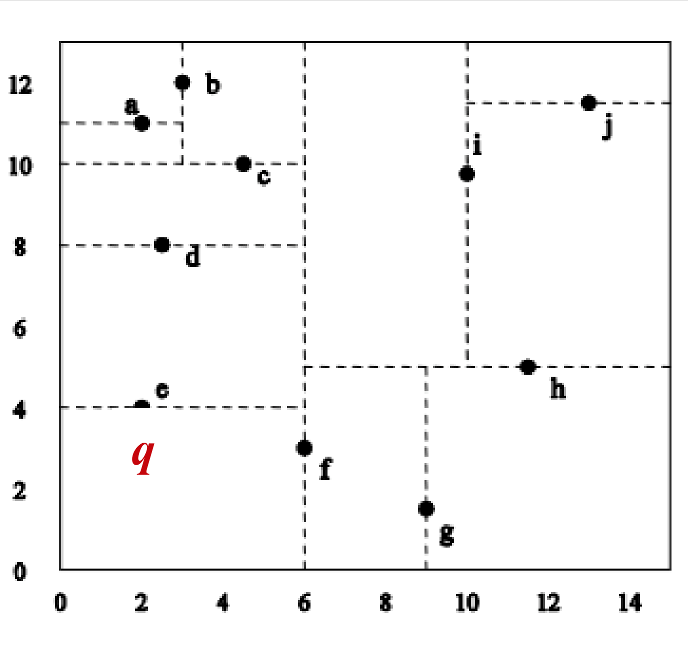

given query
$x^{(q)} = (2, 3)$



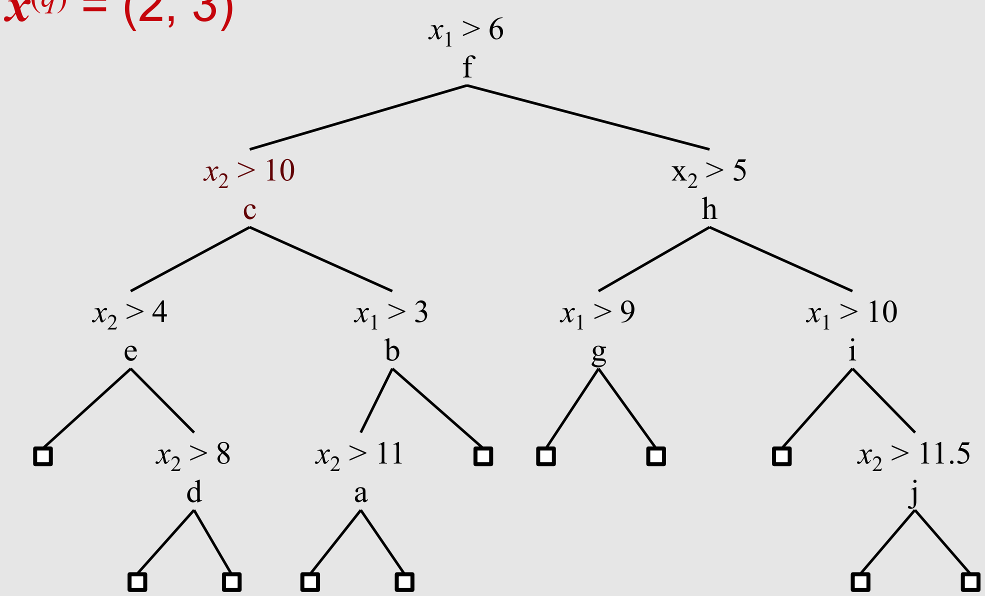

| distance | best distance | best node | priority queue |
|---|---|---|---|
| | ∞ | | (f, 0) |
| 4.0 | 4.0 | f | |
| | | | |
| | | | |

pop f

# k-d tree example (Manhattan distance)

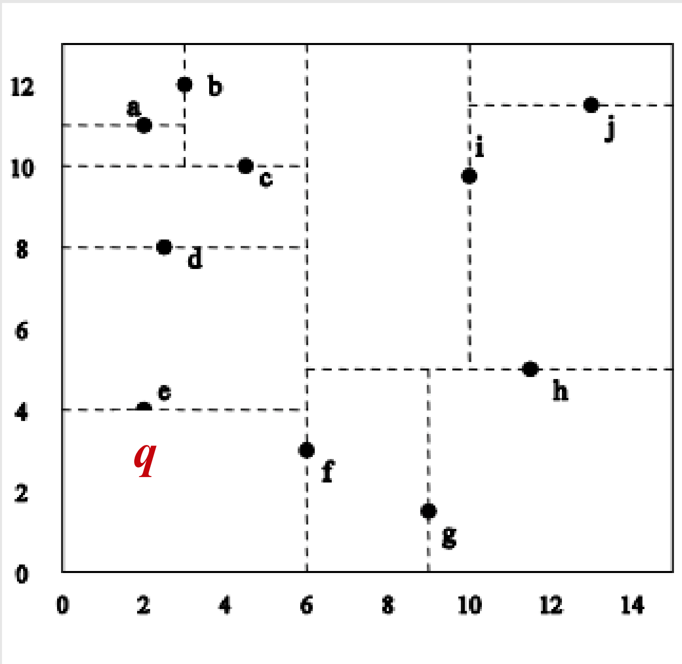



given query
$\boldsymbol{x}^{(q)} = (2, 3)$

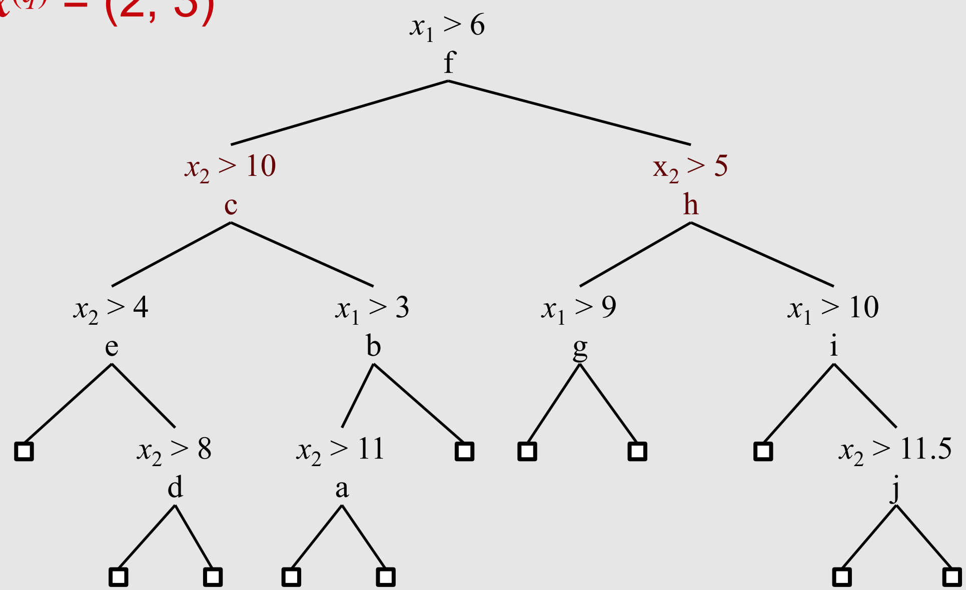



| | distance | best distance | best node | priority queue |
|---|---|---|---|---|
| | | ∞ | | (f, 0) |
| pop f | 4.0 | 4.0 | f | (c, 0) |
| | | | | |
| | | | | |

# k-d tree example (Manhattan distance)

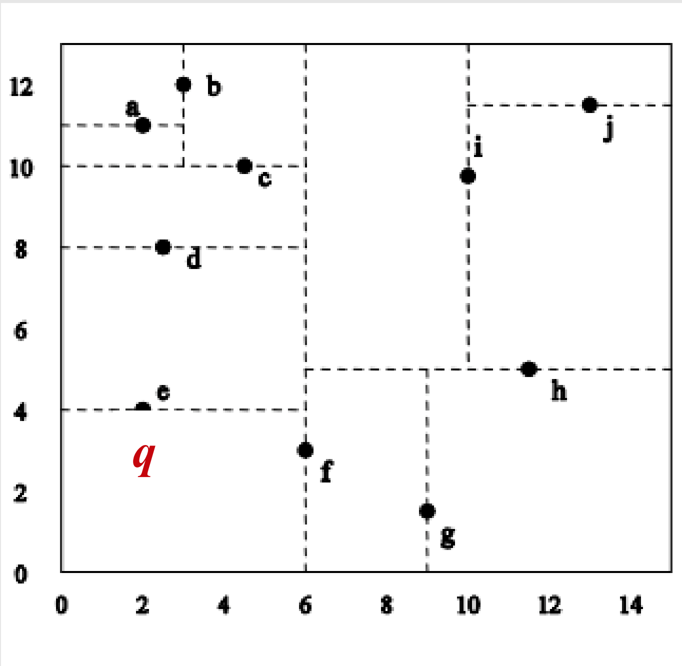



given query
$x^{(q)}$ = (2, 3)



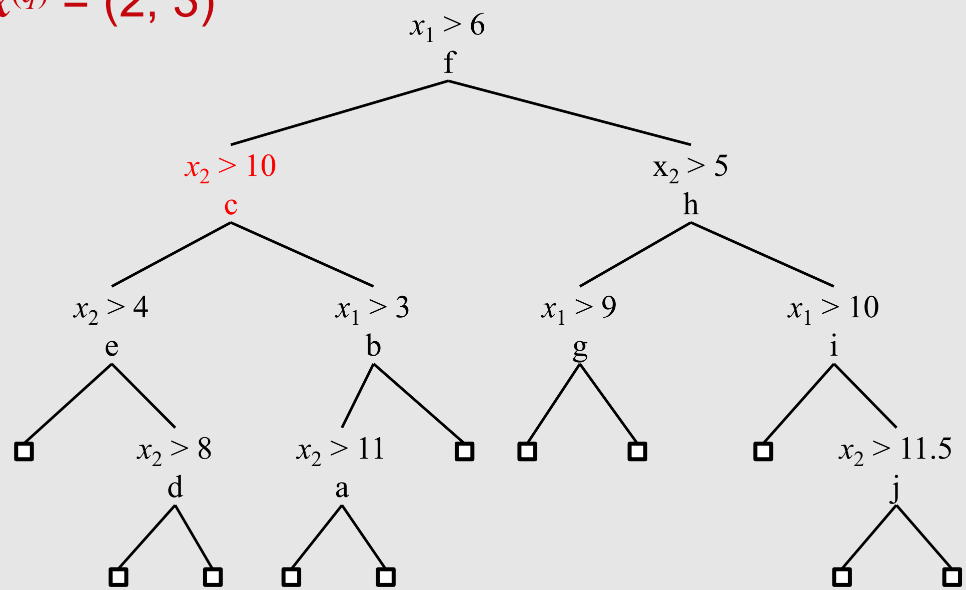

| | distance | best distance | best node | priority queue |
|---|---|---|---|---|
| | | ∞ | | (f, 0) |
| pop f | 4.0 | 4.0 | f | (c, 0) (h, 4) |
| | | | | |
| | | | | |

# k-d tree example (Manhattan distance)




given query

$\boldsymbol{x}^{(q)}$ = (2, 3)

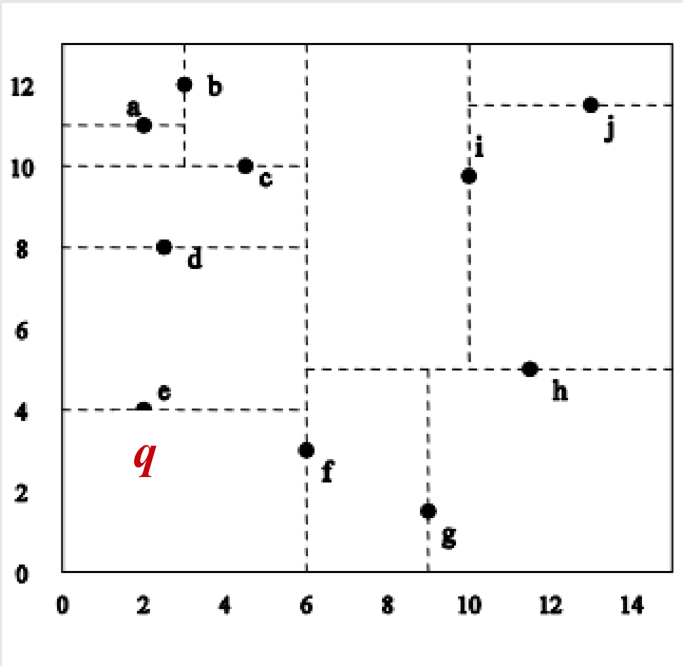


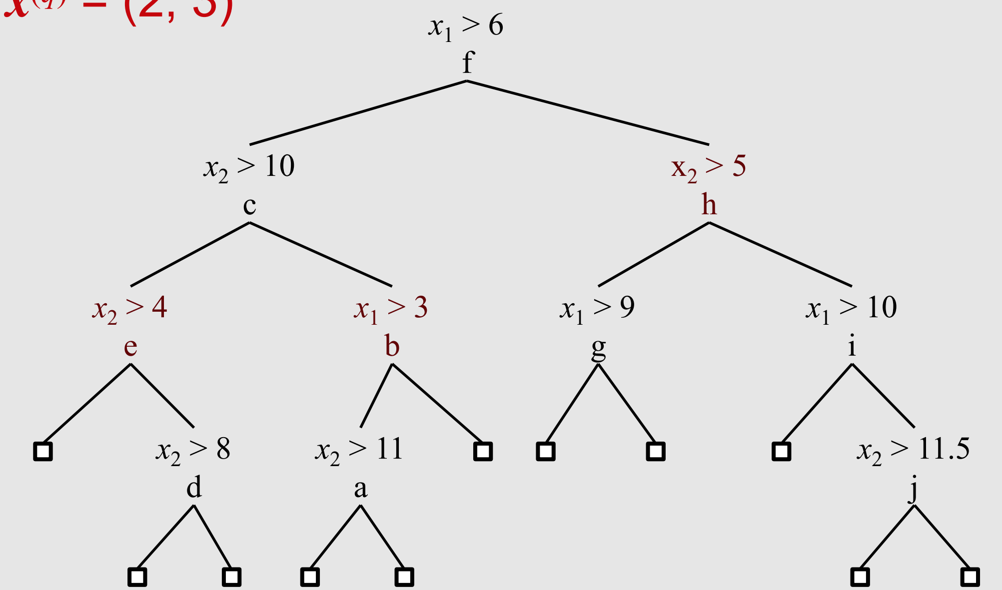

| | distance | best distance | best node | priority queue |
|---|---|---|---|---|
| | | ∞ | | (f, 0) |
| pop f | 4.0 | 4.0 | f | (c, 0) (h, 4) |
| pop c | 10.0 | 4.0 | f | |
| | | | | |

# k-d tree example (Manhattan distance)

given query

$x^{(q)}$ = (2, 3)

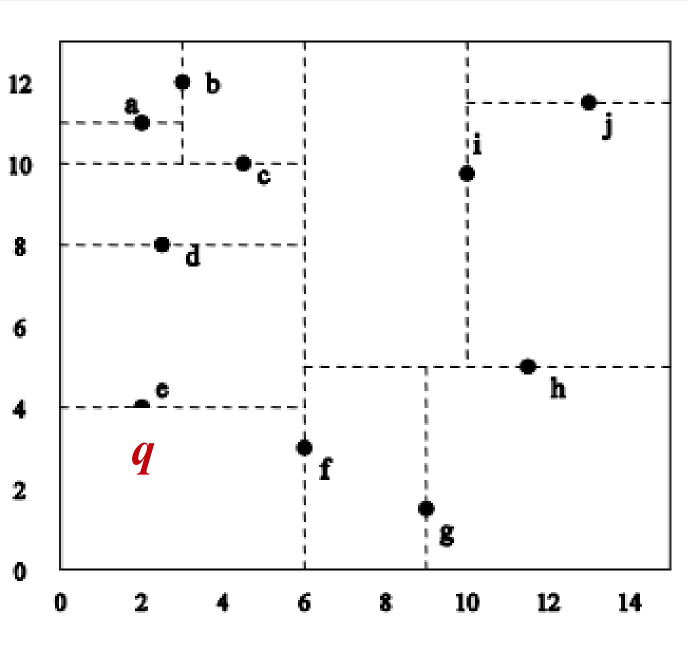


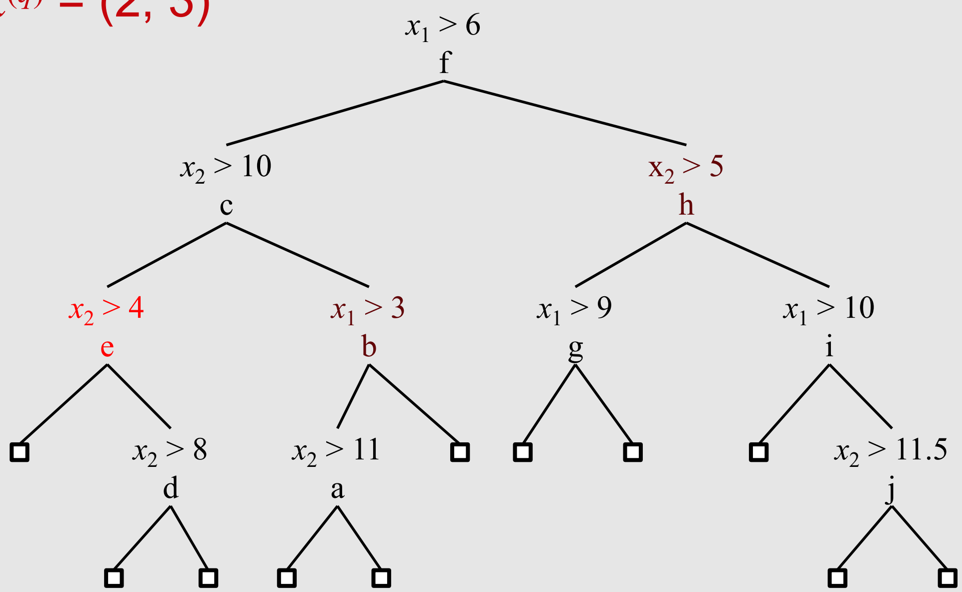




| | distance | best distance | best node | priority queue |
|---|---|---|---|---|
| | | ∞ | | (f, 0) |
| pop f | 4.0 | 4.0 | f | (c, 0) (h, 4) |
| pop c | 10.0 | 4.0 | f | (e, 0) (h, 4) (b, 7) |
| | | | | |

# k-d tree example (Manhattan distance)

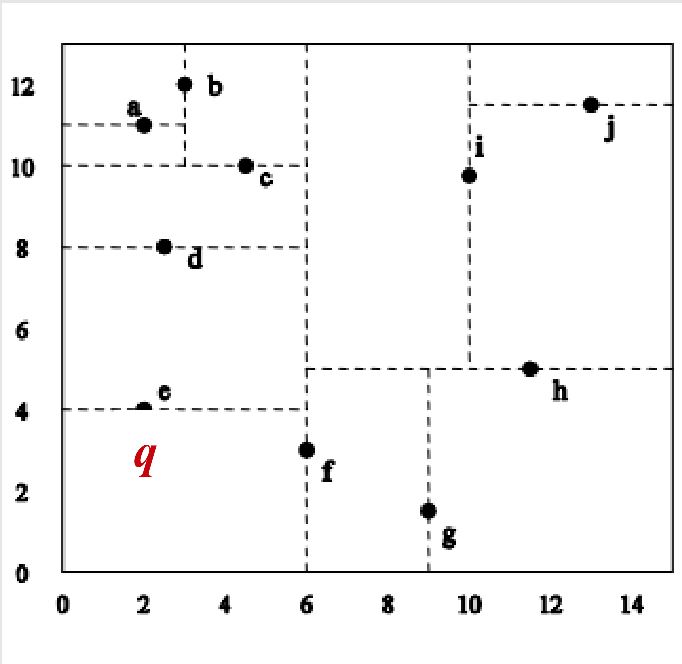
given query
$x^{(q)} = (2, 3)$



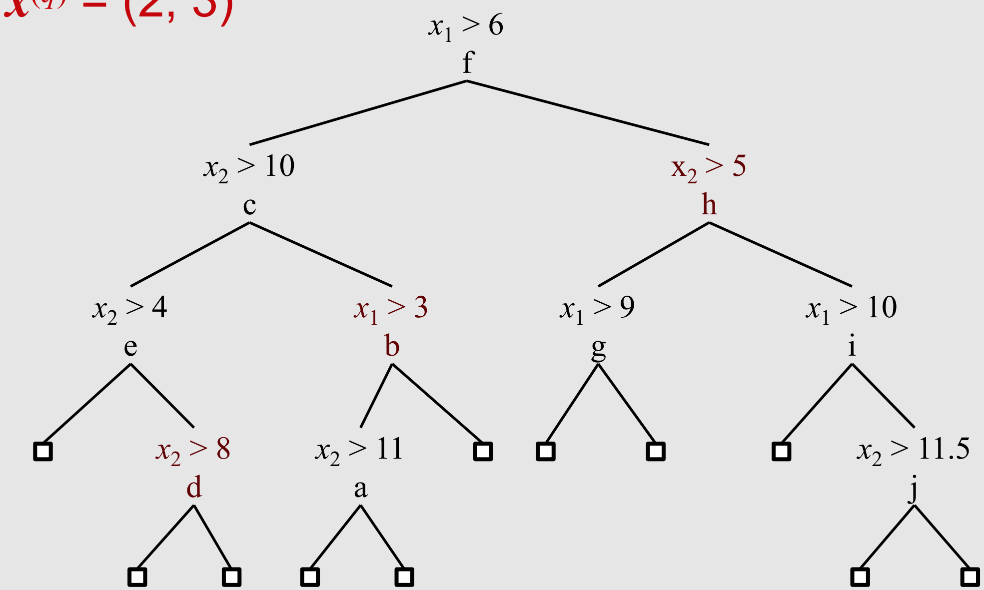




| | distance | best distance | best node | priority queue |
|---|---|---|---|---|
| | | ∞ | | (f, 0) |
| pop f | 4.0 | 4.0 | f | (c, 0) (h, 4) |
| pop c | 10.0 | 4.0 | f | (e, 0) (h, 4) (b, 7) |
| pop e | 1.0 | 1.0 | e | |

# k-d tree example (Manhattan distance)

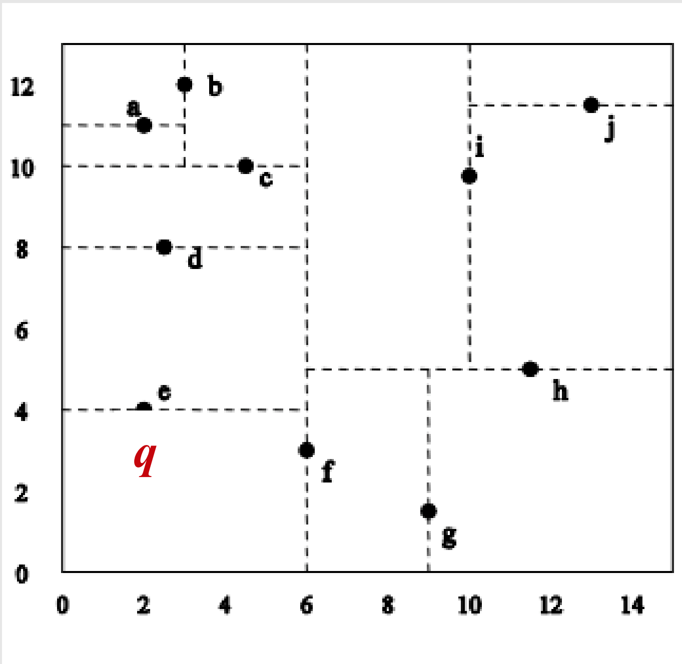



given query
$x^{(q)} = (2, 3)$

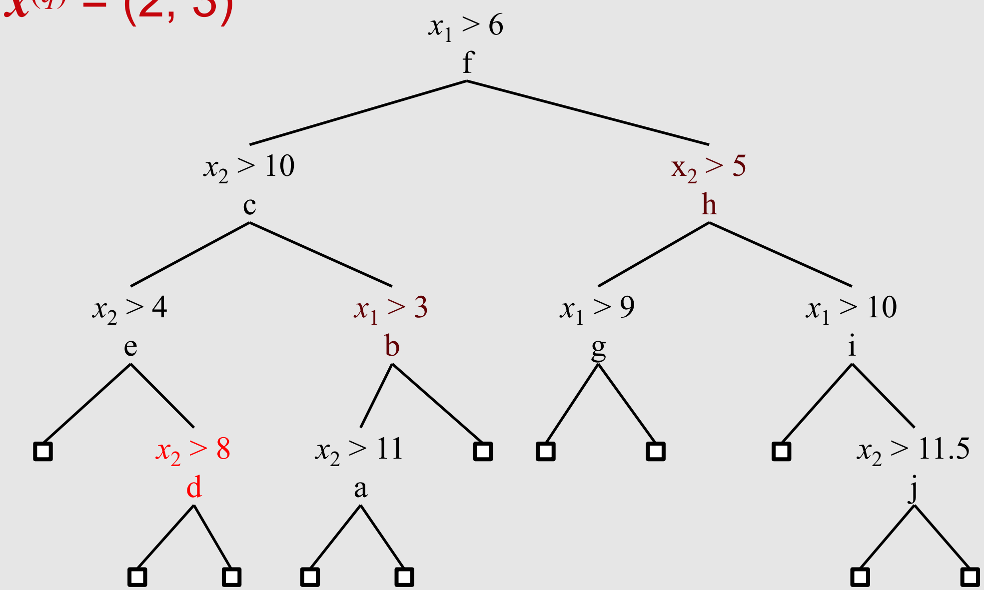



| | distance | best distance | best node | priority queue |
|---|---|---|---|---|
| | | ∞ | | (f, 0) |
| pop f | 4.0 | 4.0 | f | (c, 0) (h, 4) |
| pop c | 10.0 | 4.0 | f | (e, 0) (h, 4) (b, 7) |
| pop e | 1.0 | 1.0 | e | (d, 1) (h, 4) (b, 7) |

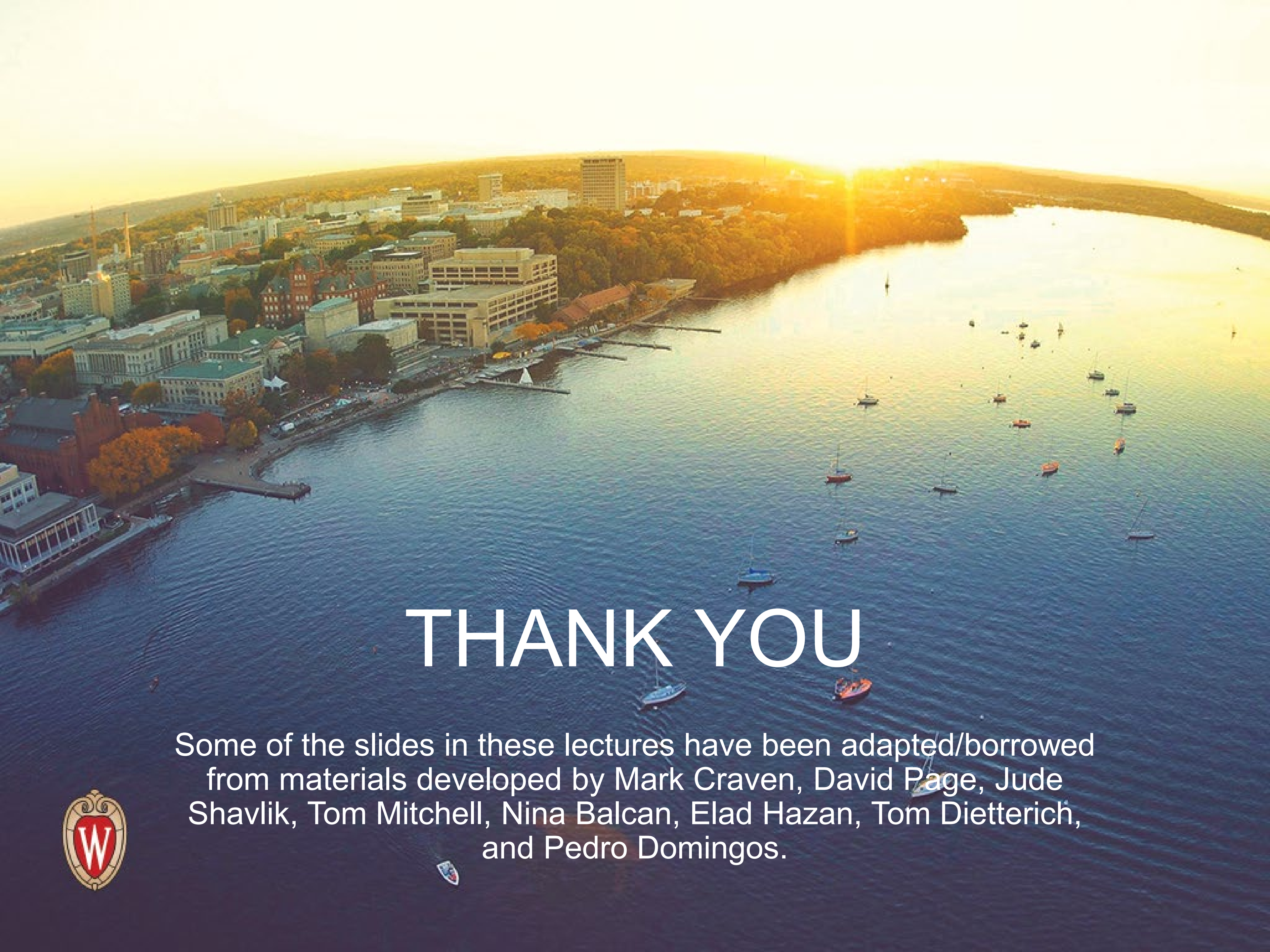
# k-d tree example (Manhattan distance)




given query
$\boldsymbol{x}^{(q)} = (2, 3)$




| | distance | best distance | best node | priority queue |
|---|---|---|---|---|
| | | ∞ | | (f, 0) |
| pop f | 4.0 | 4.0 | f | (c, 0) (h, 4) |
| pop c | 10.0 | 4.0 | f | (e, 0) (h, 4) (b, 7) |
| pop e | 1.0 | 1.0 | e | (d, 1) (h, 4) (b, 7) |
| pop d | return e | | | |


THANK YOU
Some of the slides in these lectures have been adapted/borrowed from materials developed by Mark Craven, David Page, Jude Shavlik, Tom Mitchell, Nina Balcan, Elad Hazan, Tom Dietterich, and Pedro Domingos.