Instance-Based Learning

www.biostat.wisc.edu/~dpage/cs760/

Goals for the lecture

you should understand the following concepts

- k-NN classification
- k-NN regression
- edited nearest neighbor
- k-d trees for nearest neighbor identification

Nearest-neighbor classification

learning task

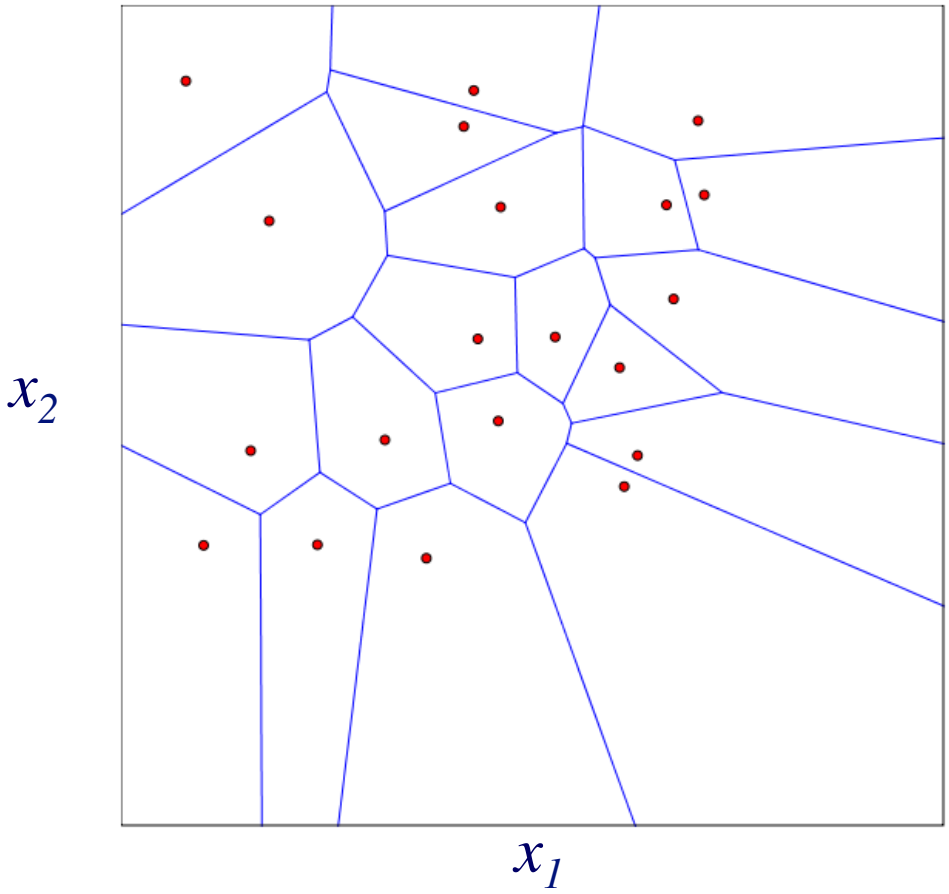
- given a training set $(\boldsymbol{x}_1, y_1) \ldots (\boldsymbol{x}_n, y_n)$, do nothing (it's sometimes called a *lazy learner*)

classification task

- **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 for nearest-neighbor classification

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

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

(i.e. return the class that the plurality of the neighbors have)

How can we determine similarity/distance

suppose all features are nominal (discrete)

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

suppose all features are continuous

- Euclidean distance:

$$d(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sqrt{\sum_f \left(x_{if} - x_{jf}\right)^2}$$ where x_{if} represents the f^{th} feature of $\boldsymbol{x}_i$

- Could also use Manhattan distance: sum the differences in feature values – continuous analog to Hamming distance

How can we determine similarity/distance

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

$$d(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sum_f \begin{cases} |x_{if} - x_{jf}| & \text{if } f \text{ is continuous} \\ 1 - \delta(x_{if}, x_{jf}) & \text{if } f \text{ is discrete} \end{cases}$$

- If all feature are of equal importance, want to apply to continuous features some type of normalization (values range 0 to 1) or standardization (values distributed according to standard normal)

k-nearest-neighbor *regression*

learning task

- given a training set $(\boldsymbol{x}_1, y_1) \ldots (\boldsymbol{x}_n, y_n)$, do nothing

prediction task

- **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:

$$\hat{y} \leftarrow \underset{v \in \text{values}(Y)}{\arg\max} \sum_{i=1}^{k} w_i \, \delta(v, y_i) \qquad w_i = \frac{1}{d(x_q, x_i)^2}$$

regression:

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

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 $(\boldsymbol{x}_i, y_i)$

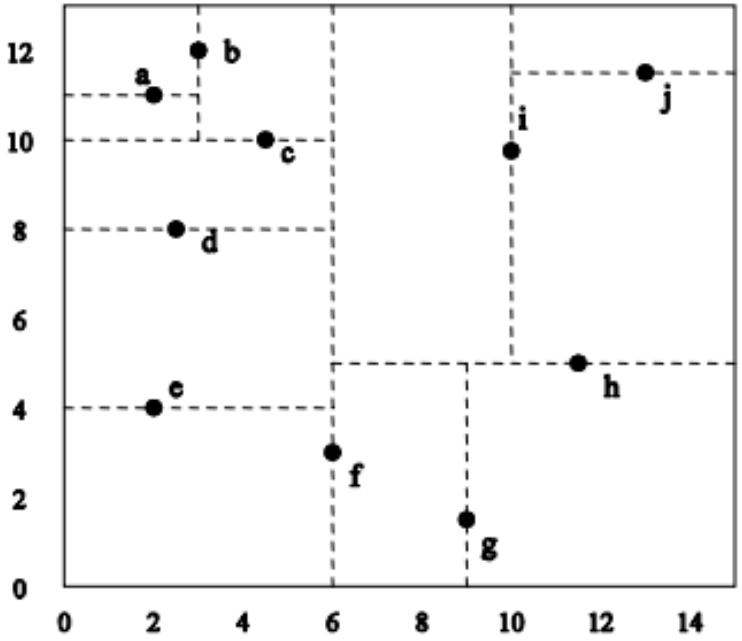
  if other training instances in memory **don't** correctly classify $(\boldsymbol{x}_i, y_i)$

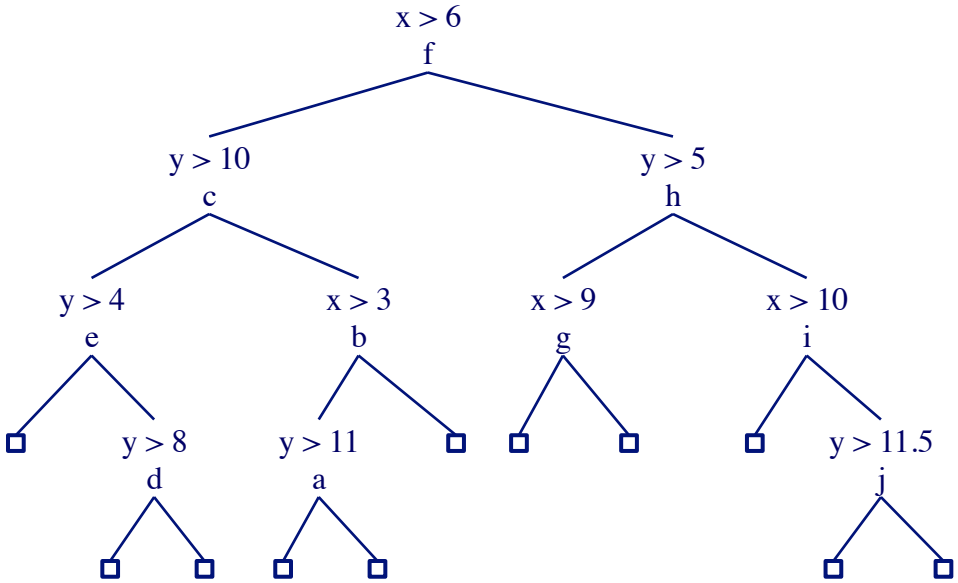
  add it to the memory

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

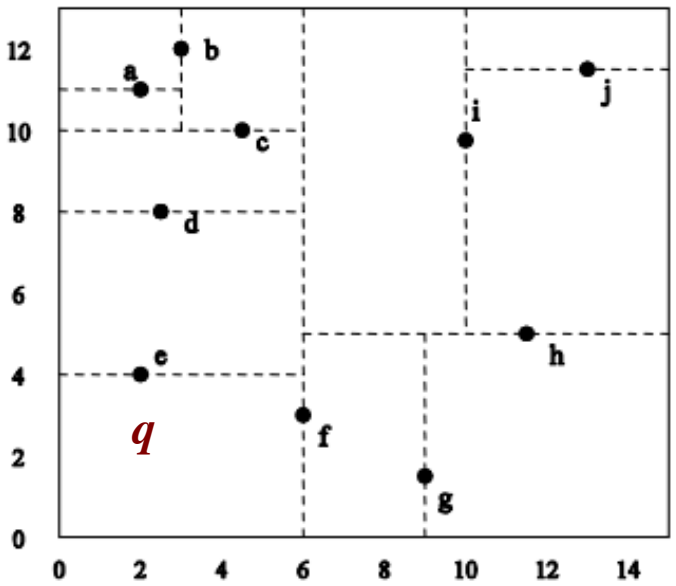
- lower bound given by distance using a single feature

- average case: $O(\log_2 n)$
- worst case: $O(n)$

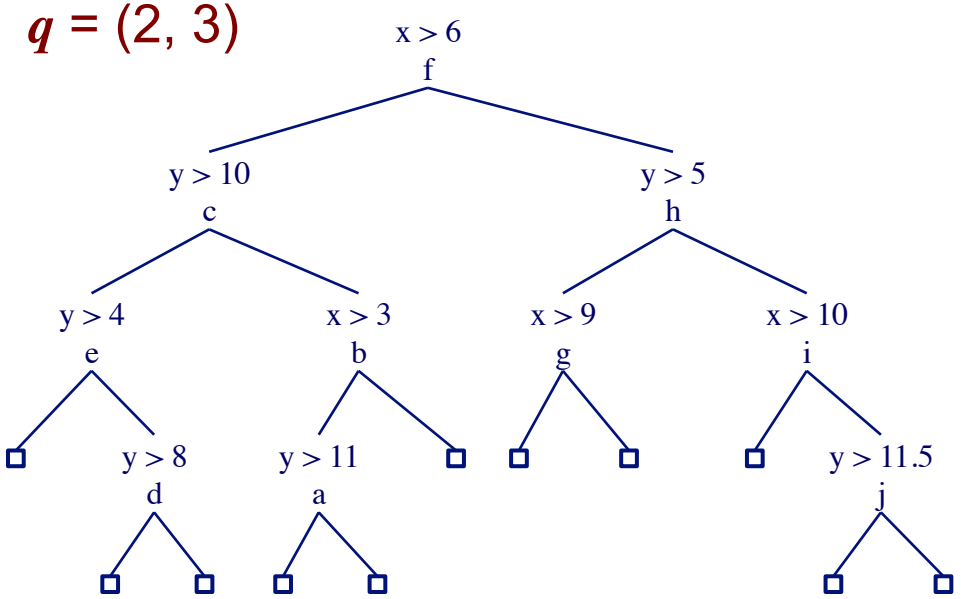
Finding nearest neighbors in a k-d tree

```
NearestNeighbor(instance 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)                        // bound is best bound on all PQ
                return best_node.instance             // so nearest neighbor found
        dist = distance(q, node. instance)
        if (dist < best_dist)
                best_dist = dist
                best_node = node
        if (q[node.feature] – node.threshold > 0)
                PQ.push(node.left, q[node.feature] – node.threshold)
                PQ.push(node.right, 0)
        else
                PQ.push(node.left, 0)
                PQ.push(node.right, node. threshold - q [node.feature])
    return best_node. instance
```

k-d tree example (Manhattan Distance)



given query
q = (2, 3)



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

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 that learn weights for different features
  - later we'll talk about *feature selection* methods
- classification 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