Instance Based Learning

Readings: CIML, Chapter 3;
Mitchell, Chapter 8;
Murphy, Chapter 1

(Slides inherited from Daniel Lowd, Vibhav Gogate, Tom Dietterich, Carlos Guestrin, Ray Mooney, Andrew Moore, Andrew Ng and others)

Some Vocabulary

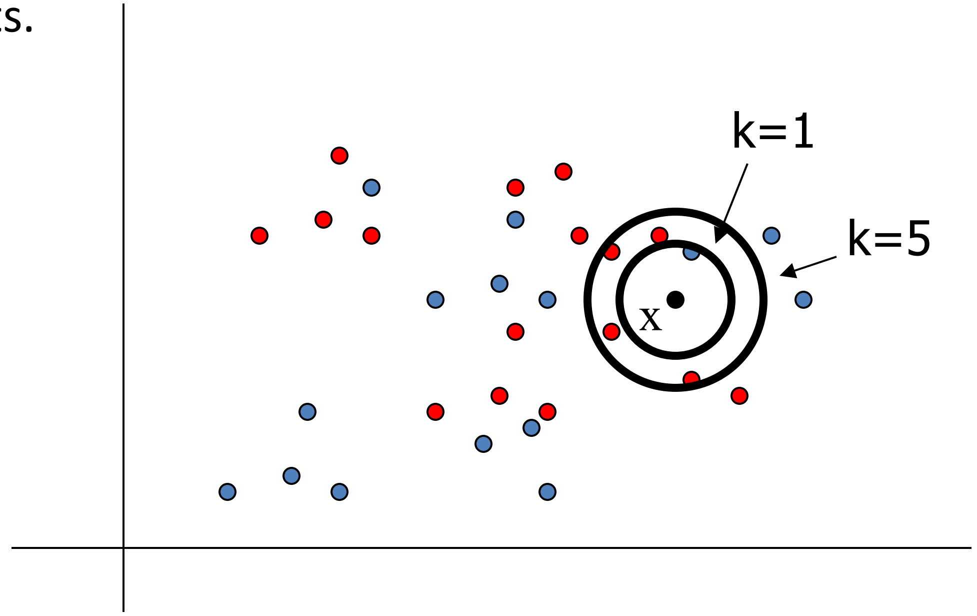
- **Parametric vs. Non-parametric:**
  - **parametric**:
    - A particular functional form is assumed (e.g., perceptron – next week!)
    - Advantage of simplicity – easy to estimate and interpret
    - may have high bias because the real data may not obey the assumed functional form.
  - **non-parametric:**
    - Distribution or density estimate is data-driven and relatively few assumptions are made a priori about the functional form.
    - <u>Data determines the model complexity</u>
- Other terms: Instance-based, Memory-based, Lazy, Case-based, kernel methods…
- Q: Is a decision tree parametric or non-parametric?

Nearest Neighbor Algorithm

- Learning Algorithm:
  - Store training examples
- Prediction Algorithm:
  - To classify a new example $\mathbf{x}$ by finding the training example $(\mathbf{x}^i, y^i)$ that is *nearest* to $\mathbf{x}$
  - Guess the class $y = y^i$

K-Nearest Neighbor Methods

- To classify a new input vector x, examine the k-closest training data points to x and assign the object to the most frequently occurring class
- Optionally, give closer points larger weights and more distant points smaller weights.

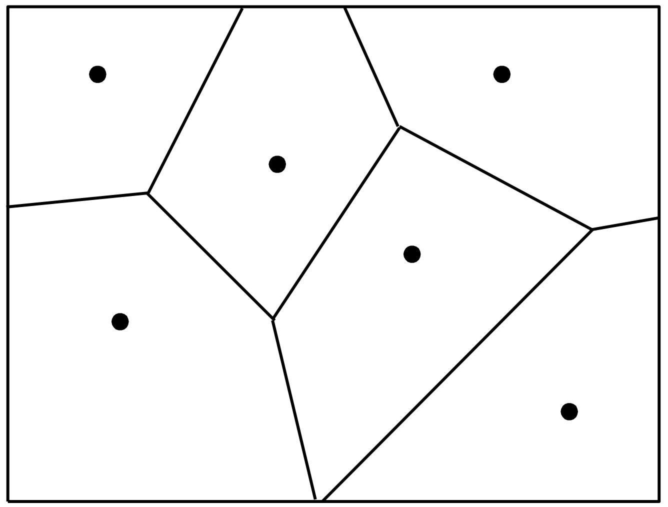


common values for k: 3, 5

Decision Boundaries

- The nearest neighbor algorithm does not explicitly compute decision boundaries. However, the decision boundaries form a subset of the Voronoi diagram for the training data.

1-NN Decision Surface



- Each line segment is equidistant between two points of opposite classes. The more examples that are stored, the more complex the decision boundaries can become.

Instance-Based Learning

Key idea: just store all training examples $< x_i, f(x_i) >$

Nearest neighbor (1-Nearest neighbor):

- Given query instance x_q, locate nearest example x_n, estimate

  $\hat{f}(x_q) \leftarrow f(x_n)$

k – Nearest neighbor:

- Given x_q, take vote among its k nearest neighbors (if discrete-valued target function)
- Take mean of f values of k nearest neighbors (if real-valued)

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

Nearest Neighbor

When to Consider

- Instances map to points in R^n
- Less than 20 attributes per instance
- Lots of training data

Advantages

- Training is very fast
- Learn complex target functions
- Do not lose information

Disadvantages

- Slow at query time (O(nd))
- Easily fooled by irrelevant attributes

Issues

- Distance measure
  - Most common: Euclidean
- Choosing k
  - Increasing k reduces variance, increases bias
  - a trade-off between overfitting (small k) and underfitting (large k)
- The curse of dimensionality: For high-dimensional space, problem that the nearest neighbor may not be very close at all!
- Memory-based technique. Must make a pass through the data for each classification. This can be prohibitive for large data sets.

Distance

- Notation: object with p measurements

$$x^i = (x^i_1, x^i_2, \ldots, x^i_p)$$

- Most common distance metric is *Euclidean* distance:

$$d_E(x^i, x^j) = \left(\sum_{k=1}^{p} (x^i_k - x^j_k)^2 \right)^{\frac{1}{2}}$$

- Efficiency trick: using squared Euclidean distance gives same answer, avoids computing square root
- ED makes sense when different measurements/features are commensurate; each is variable measured in the same units.
- If the measurements are different, say length and weight, it is not clear.

Wait: how do you convert examples to vectors?

Standardization

When variables are not commensurate, we can standardize them by dividing by the sample standard deviation. This makes them all equally important.

The estimate for the standard deviation of x_k :

$$\hat{\sigma}_k = \left(\frac{1}{n} \sum_{i=1}^{n} \left(x_k^i - \bar{x}_k \right)^2 \right)^{\frac{1}{2}}$$

where x_k is the sample mean:

$$\bar{x}_k = \frac{1}{n} \sum_{i=1}^{n} x_k^i$$

Weighted Euclidean distance

Finally, if we have some idea of the relative importance of each variable, we can weight them:

$$d_{WE}(i,j) = \left(\sum_{k=1}^{p} w_k (x_k^i - x_k^j)^2 \right)^{\frac{1}{2}}$$

One option: weight each feature by its mutual information with the class.

Other Distance Metrics

- Minkowski or L_λ metric:

$$d(i,j) = \left(\sum_{k=1}^{p} (x_k(i) - x_k(j))^{\lambda} \right)^{\frac{1}{\lambda}}$$

- Manhattan, city block or L_1 metric:

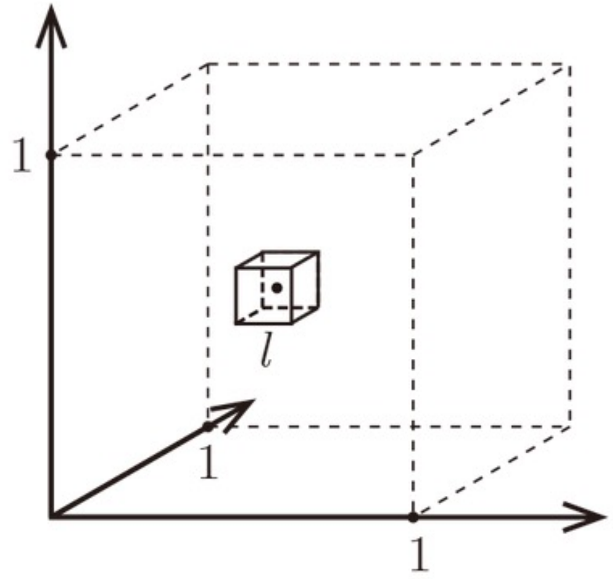
$$d(i,j) = \sum_{k=1}^{p} \left| x_k(i) - x_k(j) \right|$$

- L_∞

$$d(i,j) = \max_{k} \left| x_k(i) - x_k(j) \right|$$

The Curse of Dimensionality

- Nearest neighbor breaks down in high-dimensional spaces because the "neighborhood" becomes very large.
- i.e., in high dimensional spaces, points that are drawn from a probability distribution, are very unlikely to be close together.
- Imagine the unit cube $[0,1]^d$. n training data points are sampled ***uniformly*** within this cube.
- Consider a test point in this cube. What is the edge length l of the smallest hyper-cube that contains all the k nearest neighbors of the test point?



The Curse of Dimensionality

- Thinking about the space occupied by the hyper-cube:

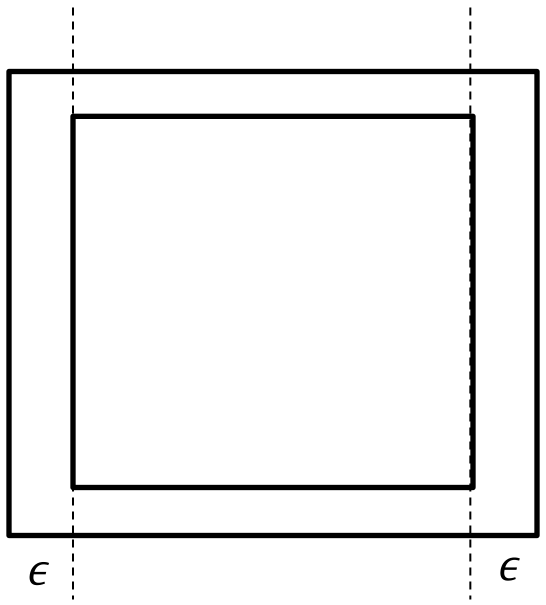
$$l^d \approx \frac{k}{n} \rightarrow l \approx \left(\frac{k}{n}\right)^{\frac{1}{d}}$$

- For $n = 1000, k = 10$:

d	l
2	0.1
10	0.63
100	0.955
1000	0.9954

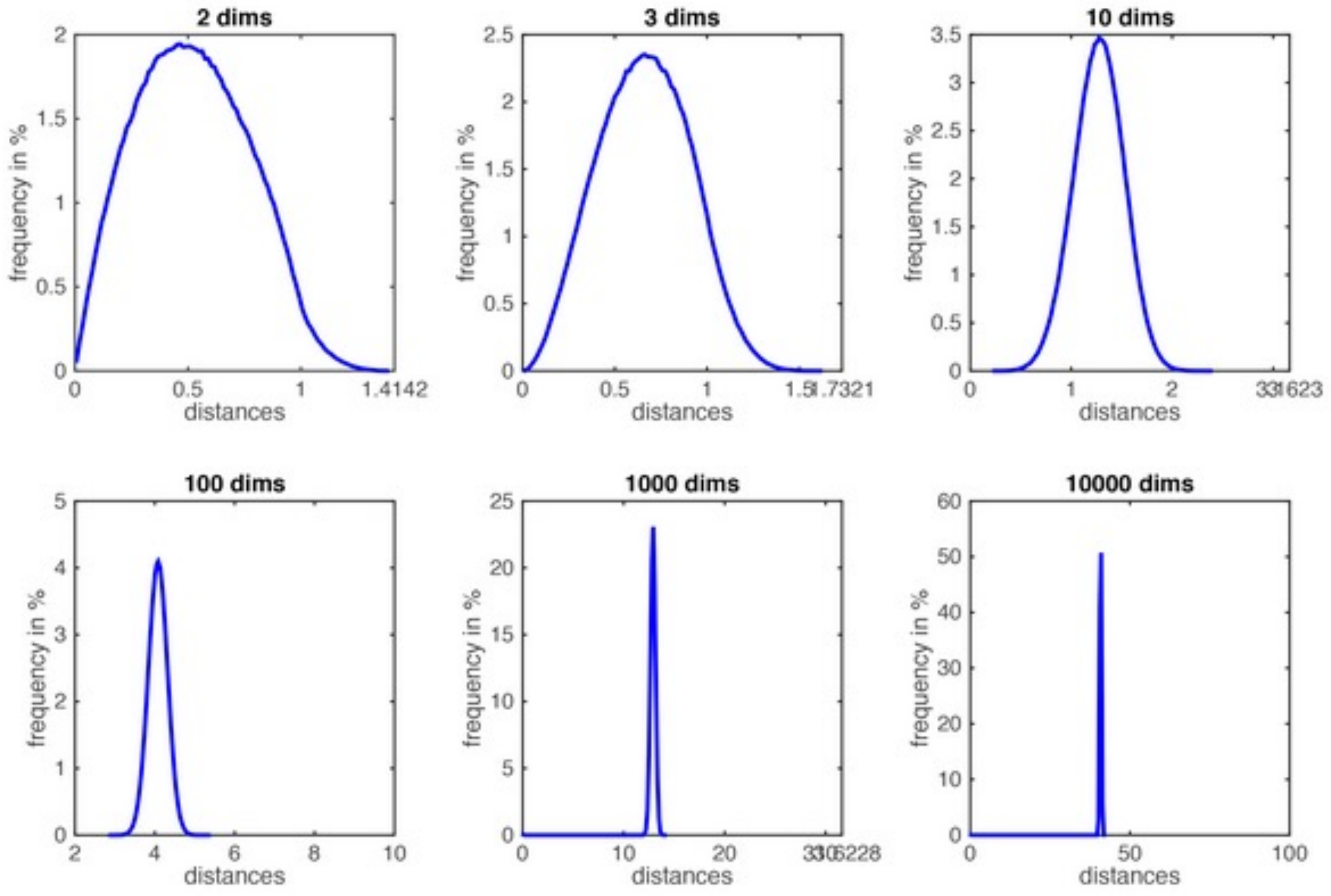
- So, as d becomes larger and larger, the randomly sampled data points tend to stay near the edges of the cube, and almost the entire space is needed to cover the k nearest neighbors.
- k nearest neighbors would break down in high dimensionality as the k nearest vectors are not particularly closer than any other data points in the training set.

The Curse of Dimensionality



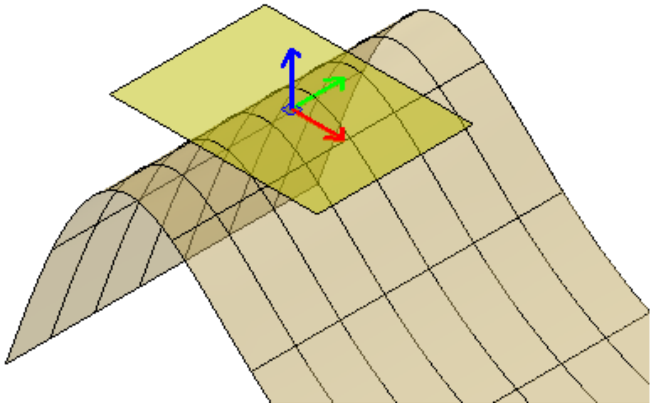
- The probability for a random point (uniformly distributed) to stay in the interior is $(1 - 2\epsilon)^d$ that becomes very small when d become very large.

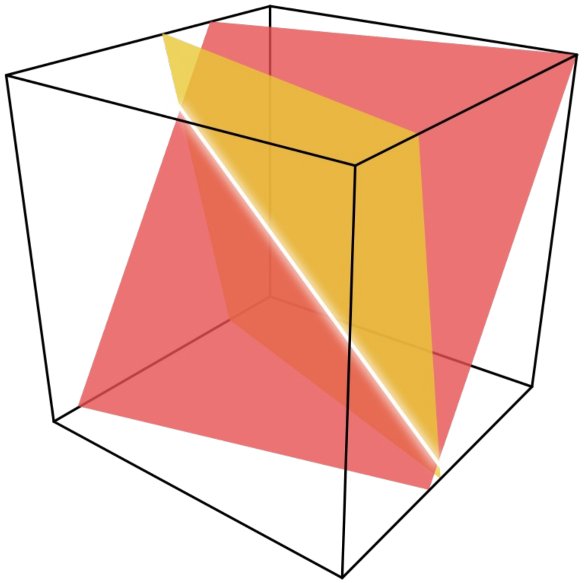
The Curse of Dimensionality



Can nearest neighbor still work in high dimensional space?

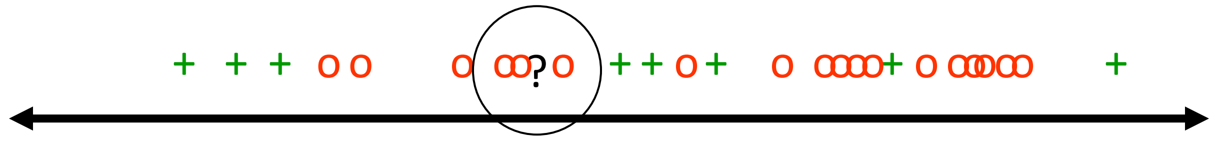
- Yes as in practice our data are not distributed uniformly in general.
- Nearest neighbor might still work if there is a low intrinsic dimensionality in the data



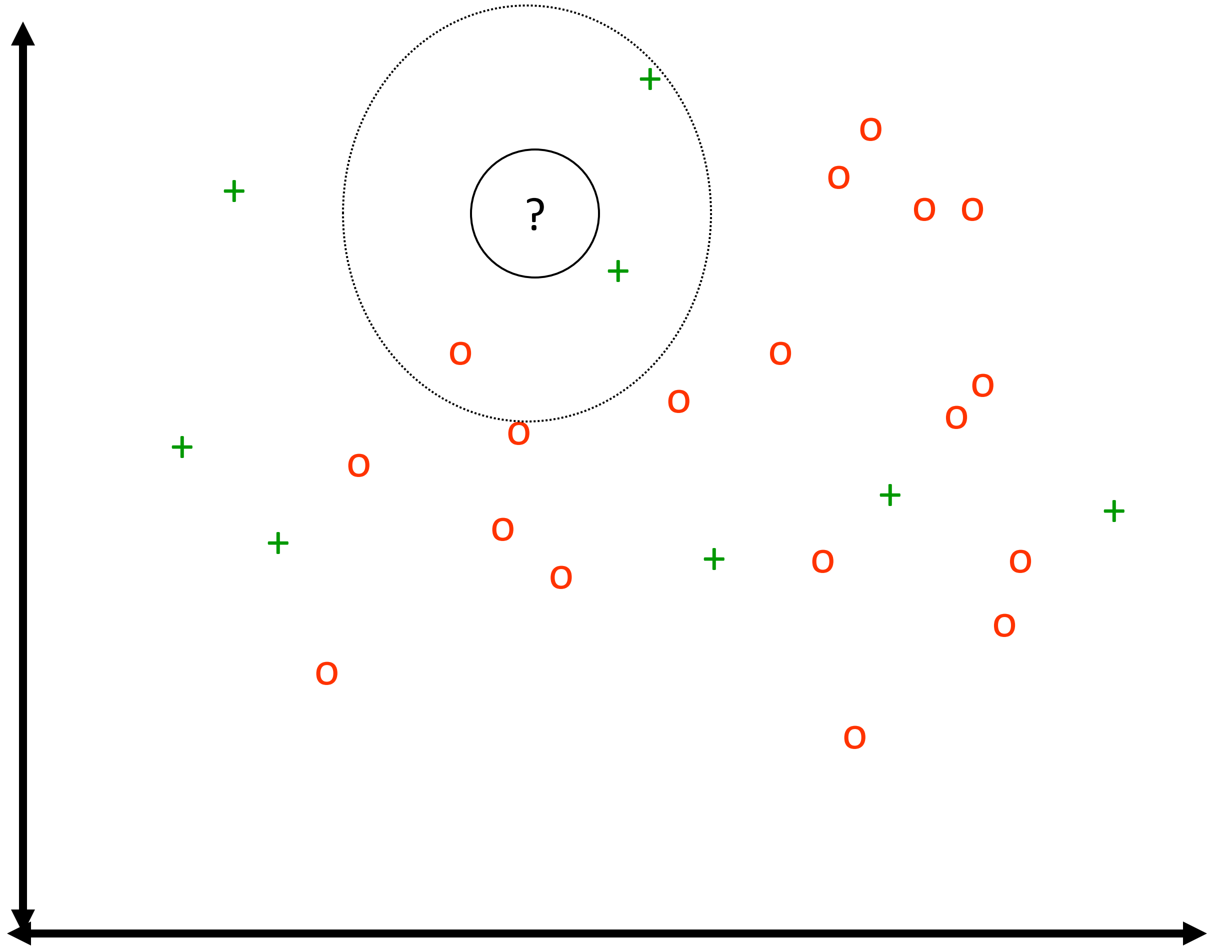


K-NN and irrelevant features

- Irrelevant features can be very harmful for k-NN

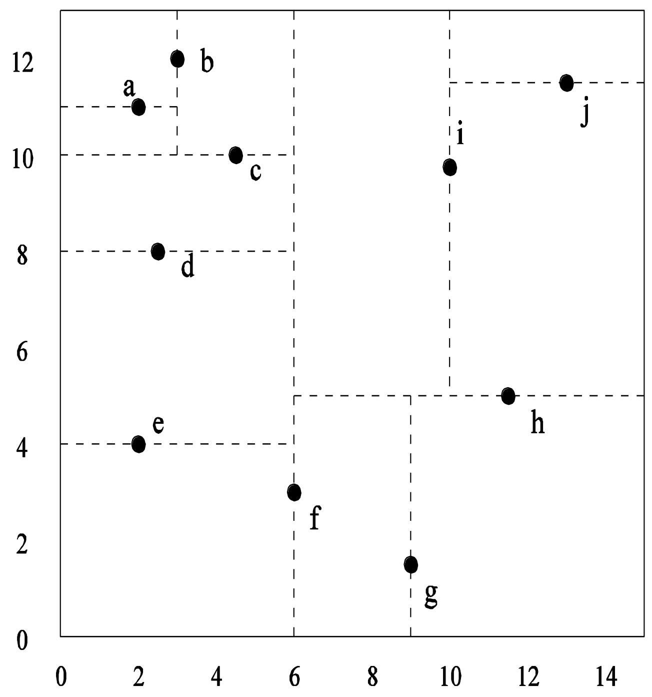


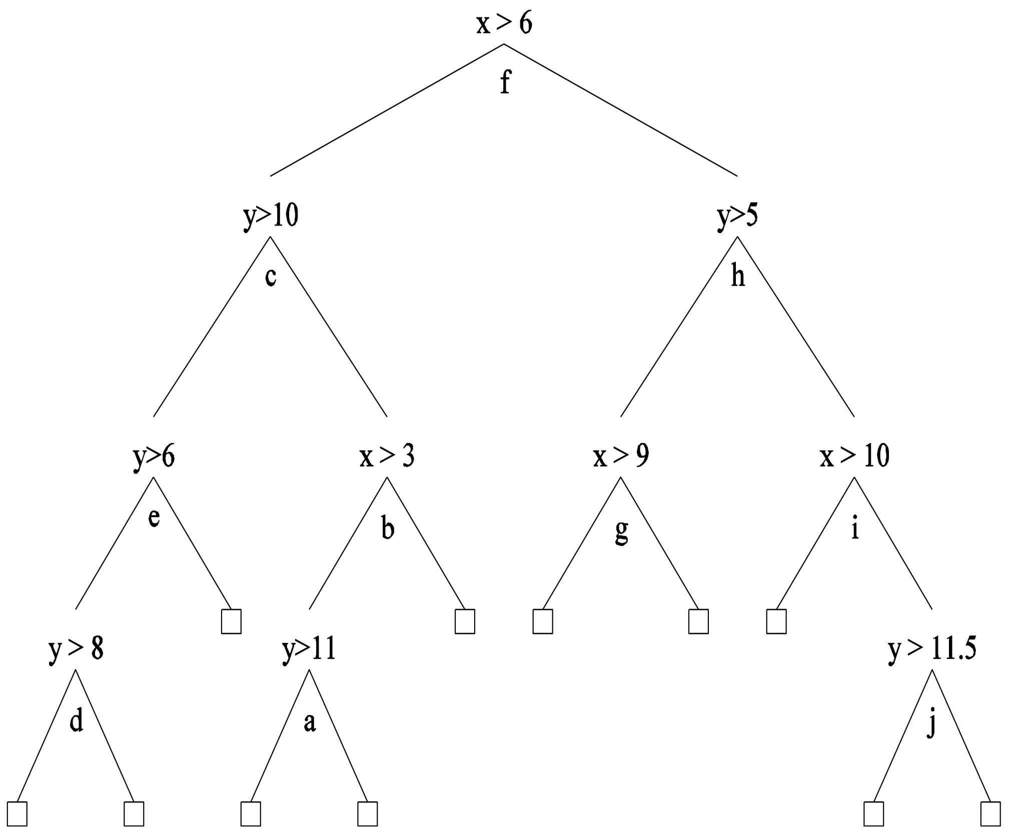
K-NN and irrelevant features



Efficient Indexing: Kd-trees

- A kd-tree is similar to a decision tree, except that we split using the median value along the dimension having the highest variance, and points are stored





Edited Nearest Neighbor

- Storing all of the training examples can require a huge amount of memory. Select a subset of points that still give good classifications.
  - **Incremental deletion**. Loop through the training data and test each point to see if it can be correctly classified given the other points. If so, delete it from the data set.
  - **Incremental growth.** Start with an empty data set. Add each point to the data set only if it is not correctly classified by the points already stored.

Nearest Neighbor Summary

- Advantages
  - Variable-sized hypothesis space
  - Learning is extremely efficient (no optimization involved)
    - However, growing a good kd-tree can be expensive
  - Very flexible decision boundaries
  - Easy to implement
  - Can be very effective
- Disadvantages
  - Distance function must be carefully chosen
  - Irrelevant or correlated features must be eliminated
  - Typically cannot handle more than 30 features
  - Computational costs: Memory and classification-time computation