Nearest Neighbors

CS 534: Machine Learning

Slides adapted from David Sontag, Luke Zettlermoyer, Carlos Guestrin, Vibhav Gogate, Mehryar Mohri, Fei Sha, Yan Liu, and Sofus A. Macskassy

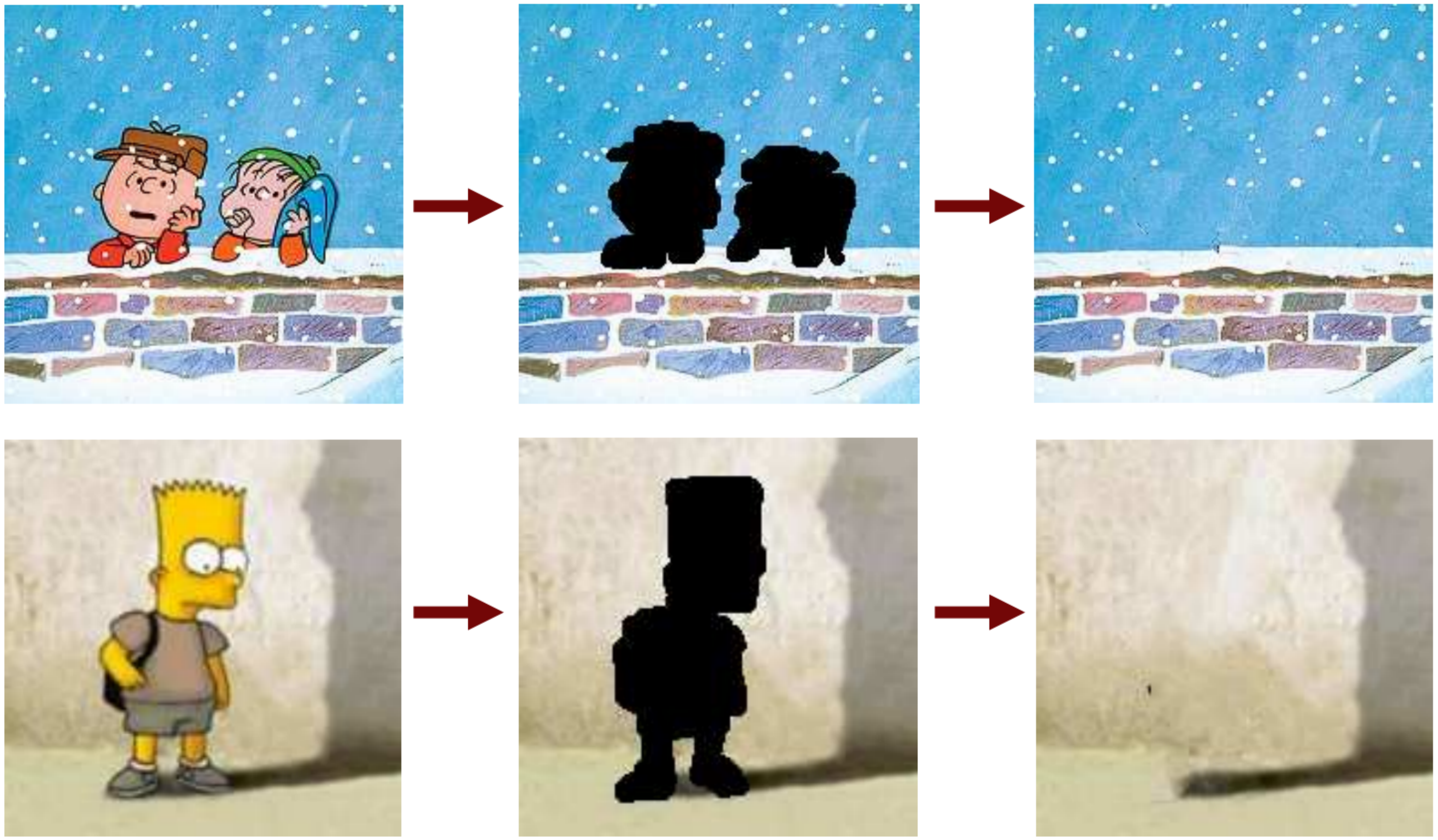
Review: Model Comparison

Criterion	Logistic Regression	LDA	Trees	SVM	Neural networks
Mixed data	no	no	yes	no	no
Missing values	no	yes	yes	no	no
Outliers	yes	no	yes	yes	yes
Robust to monotone feature transformation	no	no	yes	no	somewhat
Scalability	yes	yes	yes	no	yes
Irrelevant inputs	no	no	somewhat	somewhat	no
Extract linear combinations	yes	yes	no	yes	yes
Interpretable	yes	yes	yes	yes	no
Accurate	yes	yes	no	yes	yes

Motivation: Similar Sets

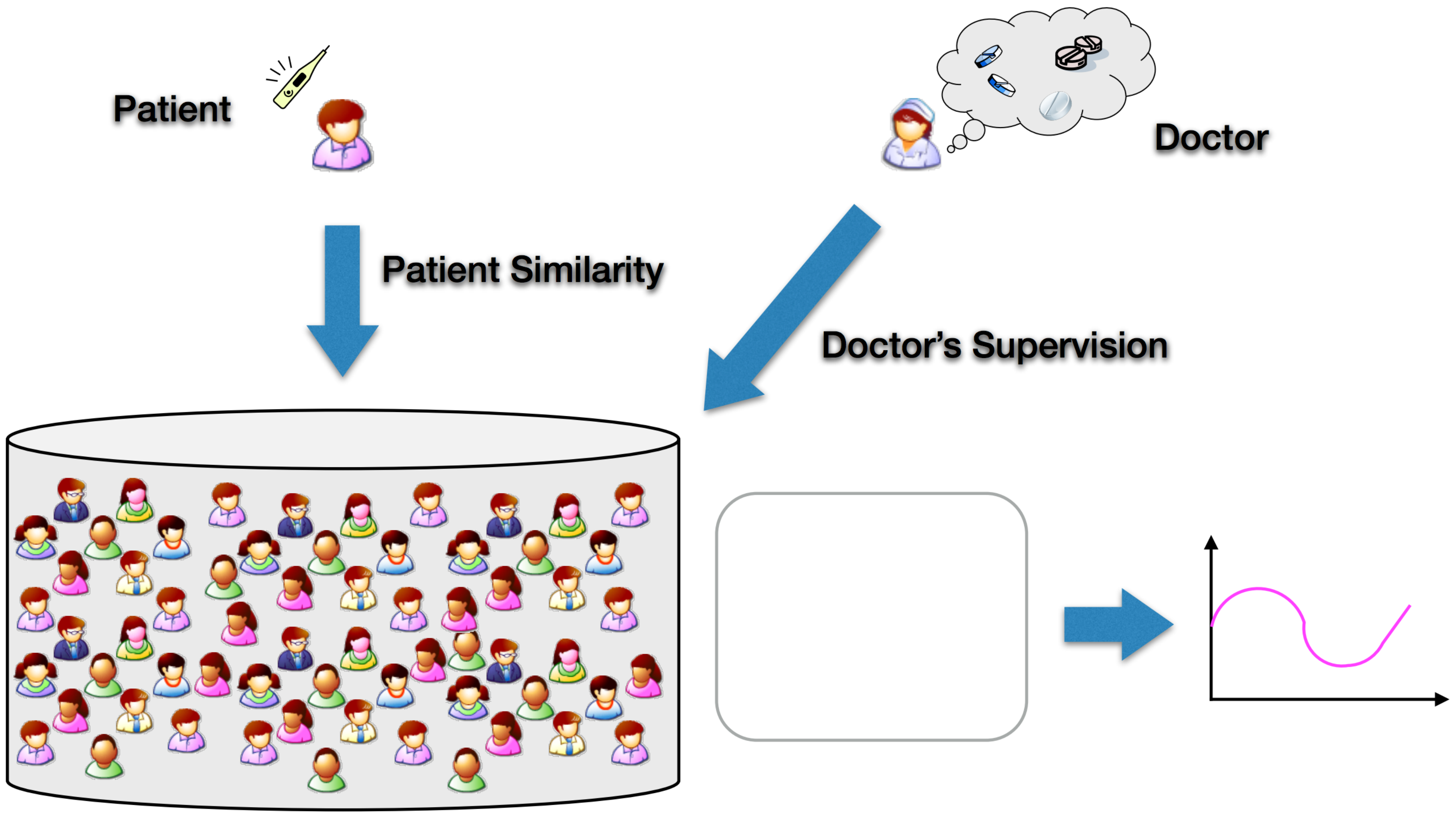
- Many problems can be expressed as finding “similar” sets
  - Data compression
  - Information retrieval
  - Pattern recognition
  - …

Applications: Image Completion



https://graphics.stanford.edu/courses/cs468-06-fall/Slide/aneesh-michael.pdf

Applications: Patient Prognosis



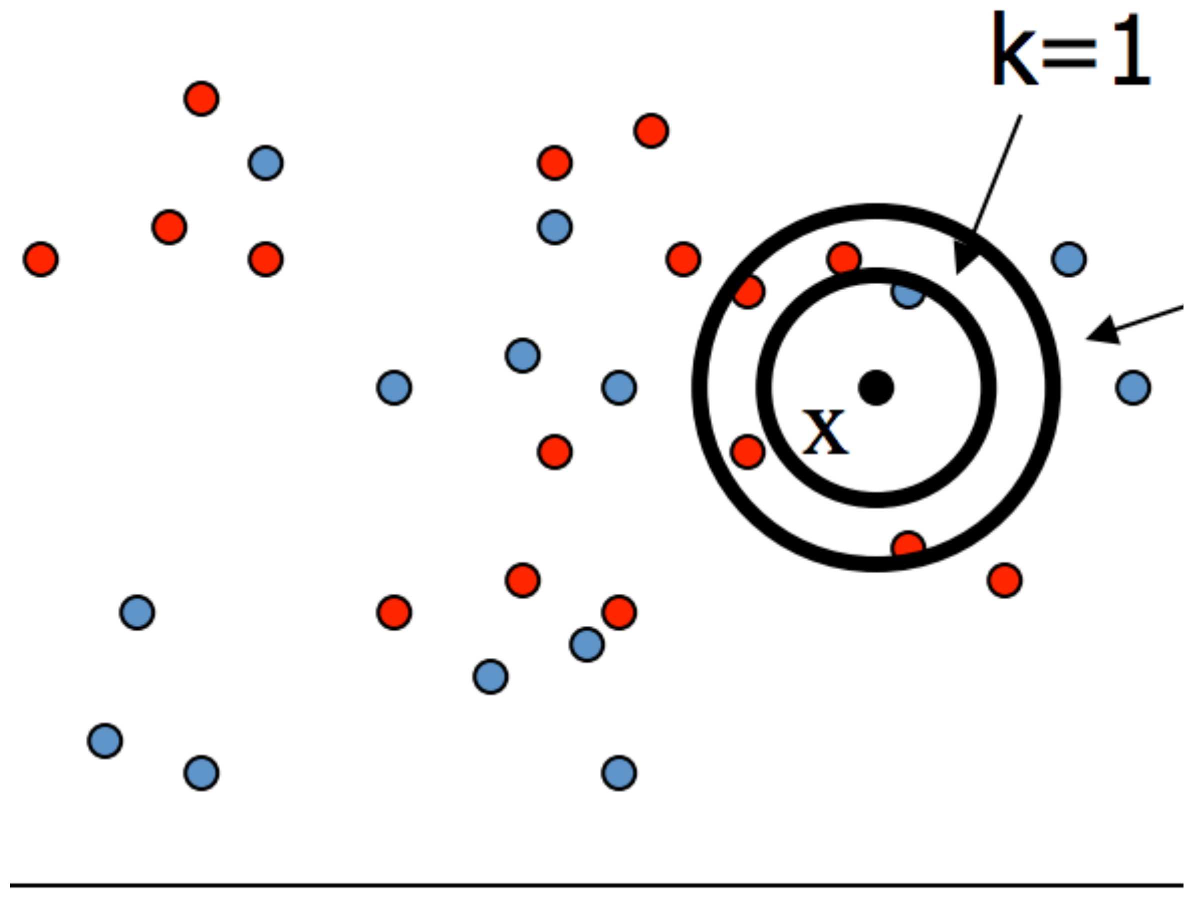
http://engr.uconn.edu/~fwang/tutorials/CIKM14_Tutorial.pdf

Nearest Neighbor (NN)

- Problem Statement: Given a set of points or samples {p_1, …, p_N}, and a new point q, find the data point nearest to q
- (AKA) Closest-point problem or post office problem
- Training samples are prototypes — no explicit model needed
- Example: http://www.theparticle.com/applets/ml/nearest_neighbor

k-NN Algorithm

- Examine the k-“closest” training data points to new point x
    - Closest depends on distance metric used
- Assign the object the most frequently occurring class (majority vote) or the average value (regression)



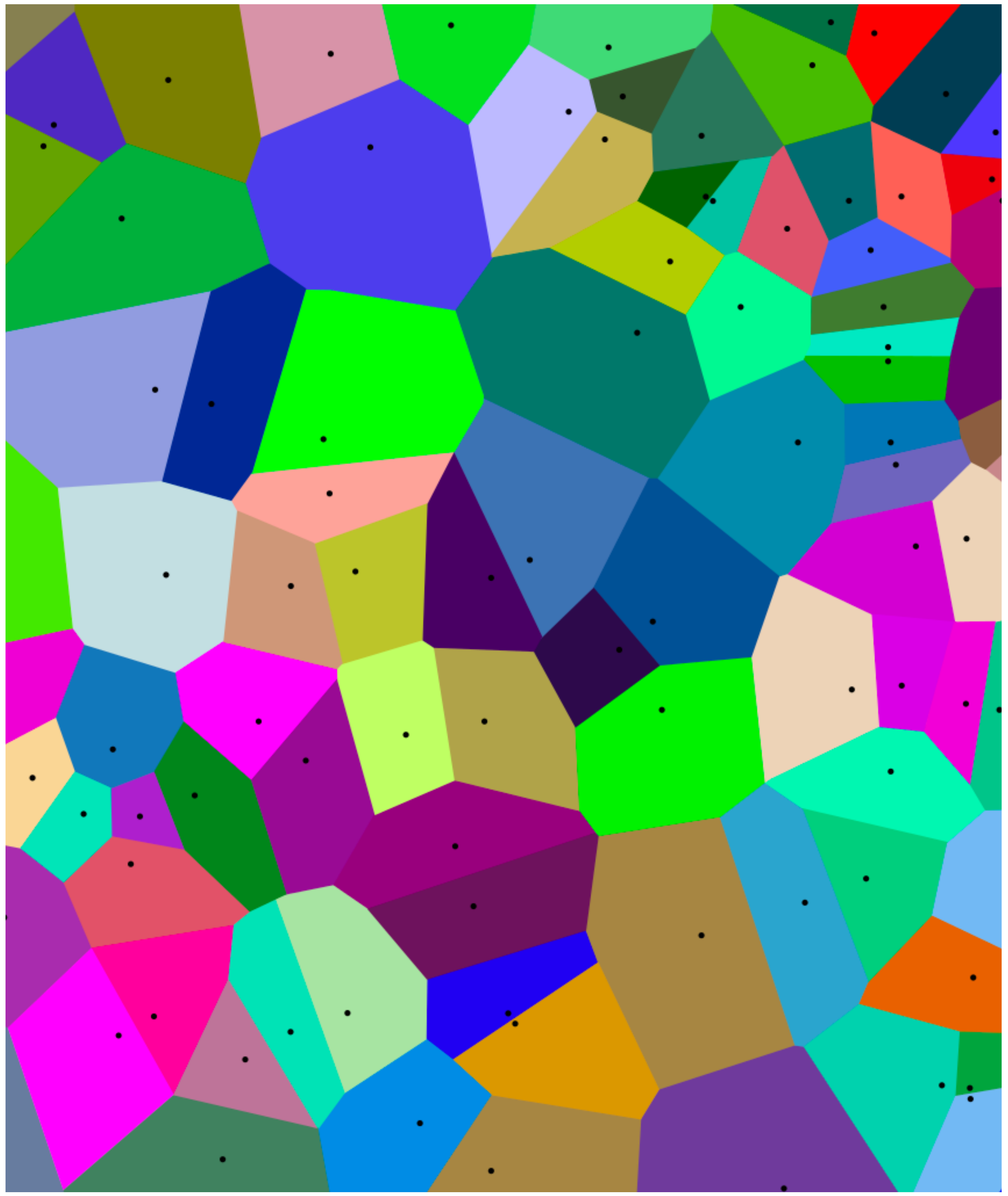
http://cs.nyu.edu/~dsontag/courses/ml13/slides/lecture11.pdf

k-NN Algorithm

- Non-parametric model — number and nature of parameters are flexible and not fixed in advance
- Lazy learner — no training needed, just store all the training examples
- Flexible decision boundaries
- One of the most popular algorithms (ranked 8th by KD Nuggets)

1-NN: Decision Boundaries

- Does not explicitly compute decision boundaries
- Decision boundaries form subset of Voronoi diagram for training data (Euclidean distance)
  - Line segment equidistant to neighboring points
- More data —> more complex decision boundaries


https://en.wikipedia.org/wiki/Voronoi_diagram

1-NN: Comparison to Bayes

- Famous result of Cover and Hart, 1967

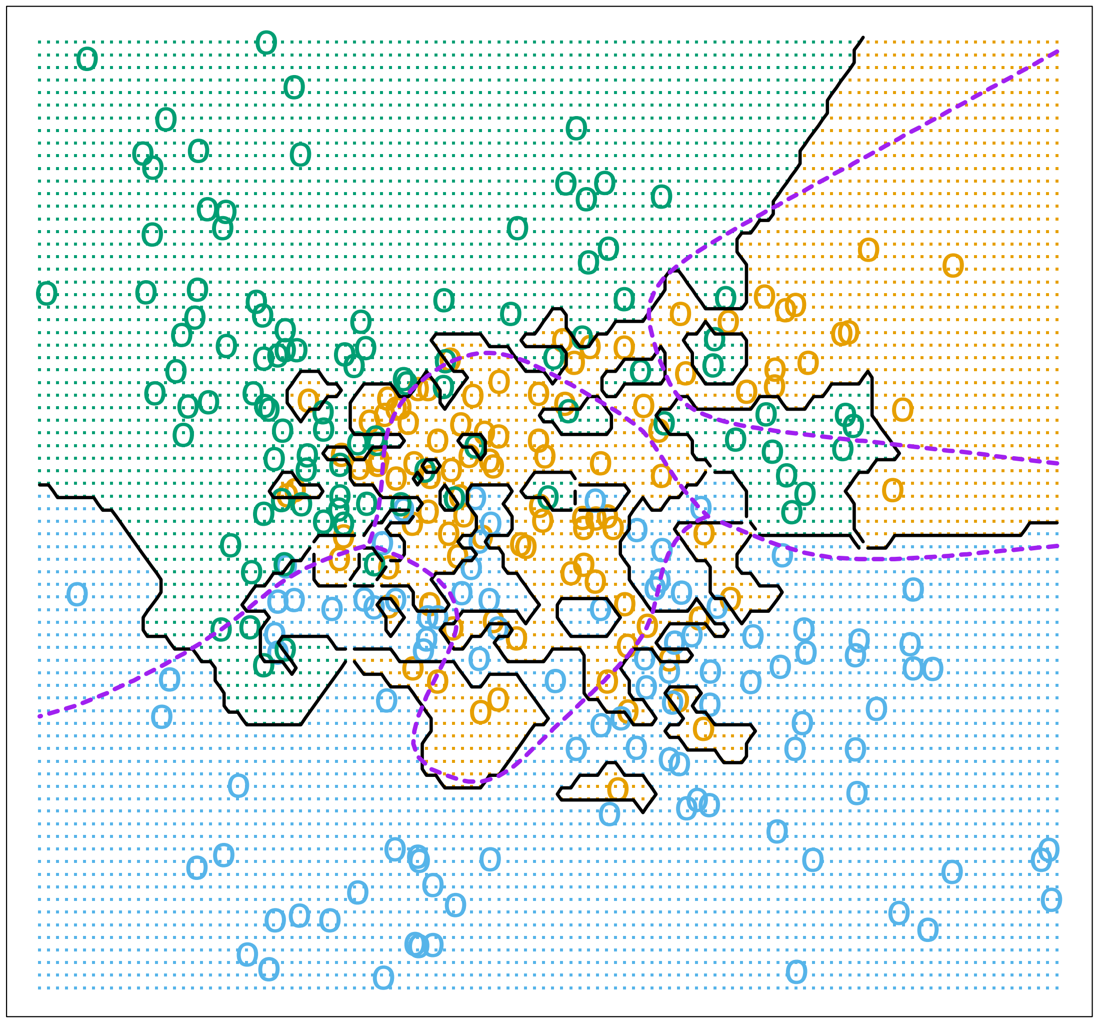
$$R(f^*) \leq R(f^{\mathrm{nn}}) \leq 2R(f^*)(1 - R(f^*)) \leq 2R(f^*)$$

- Expected risk of NN is at worst twice that of the Bayes optimal classifier

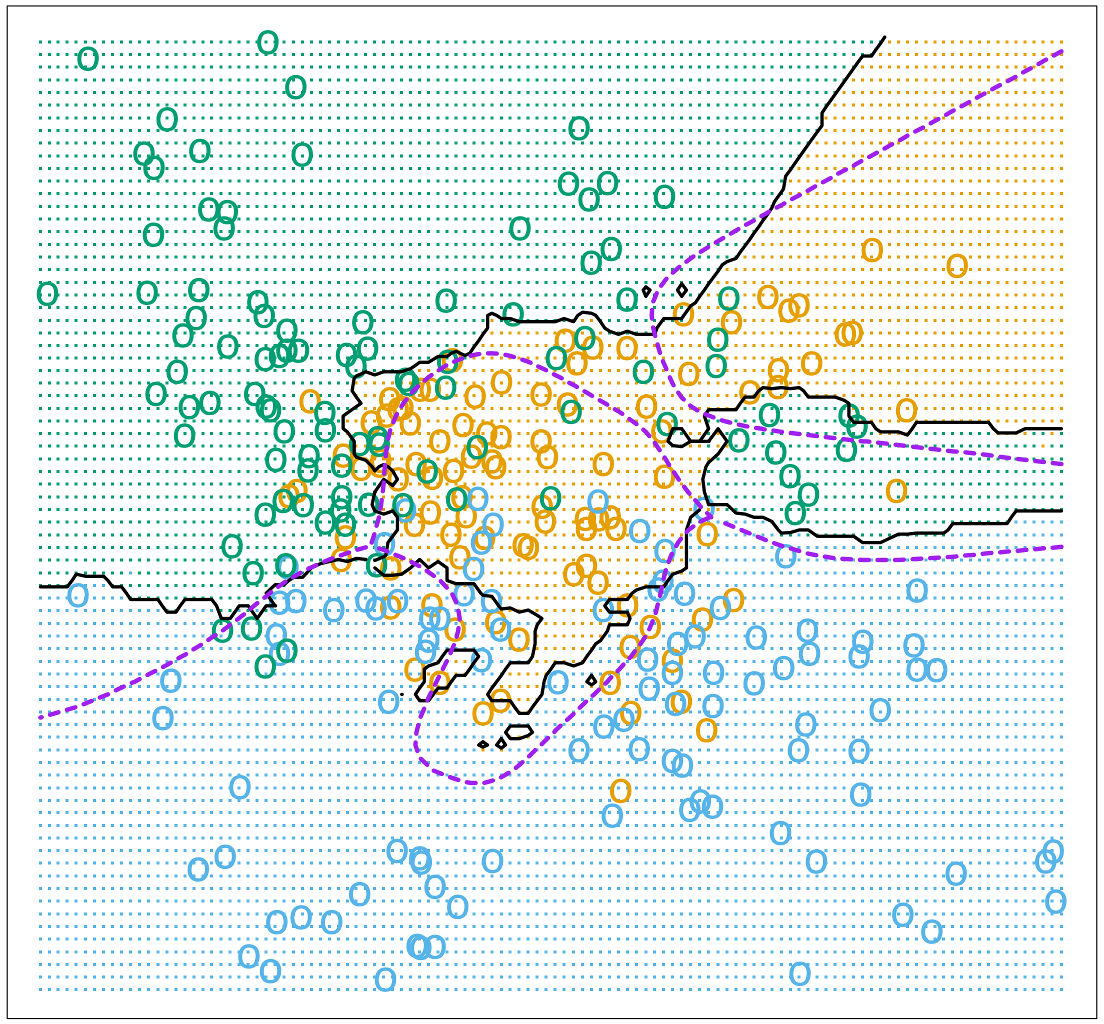
- Nearest neighbor classifier seems to be doing the right thing

k-NN: Smoothing

1-Nearest Neighbor



15-Nearest Neighbors



k-NN: Bias & Variance Intuition

- Low values of k: local model —> complex boundaries
  - Bias = low, variance = high
- High values of k: global model —> smooth boundaries
  - Bias = high, variance = low

k-NN: Bias & Variance

- Analyze using regression fit to understand

$$\begin{aligned}\mathrm{EPE}_k(\mathbf{x}_0) &= E[(Y - \hat{f}_k(\mathbf{x}_0)^2 | \mathbf{x} = \mathbf{x}_0] \\ &= \sigma^2 + [f(\mathbf{x}_0) - \frac{1}{k}\sum_j f(\mathbf{x}_{(j)})]^2 + \frac{\sigma^2}{k}\end{aligned}$$

- Higher k: variance decreases but bias increases
- Lower k: bias decreases but variance increases

k-NN: Practical Challenges

- How to pick k?
    - Small k —> noisy estimates
    - Large k —> smoothing may hurt accuracy
- What is the right measure of closeness?
    - Euclidean distance? $||x - y||_2$

k-NN: Choice of k

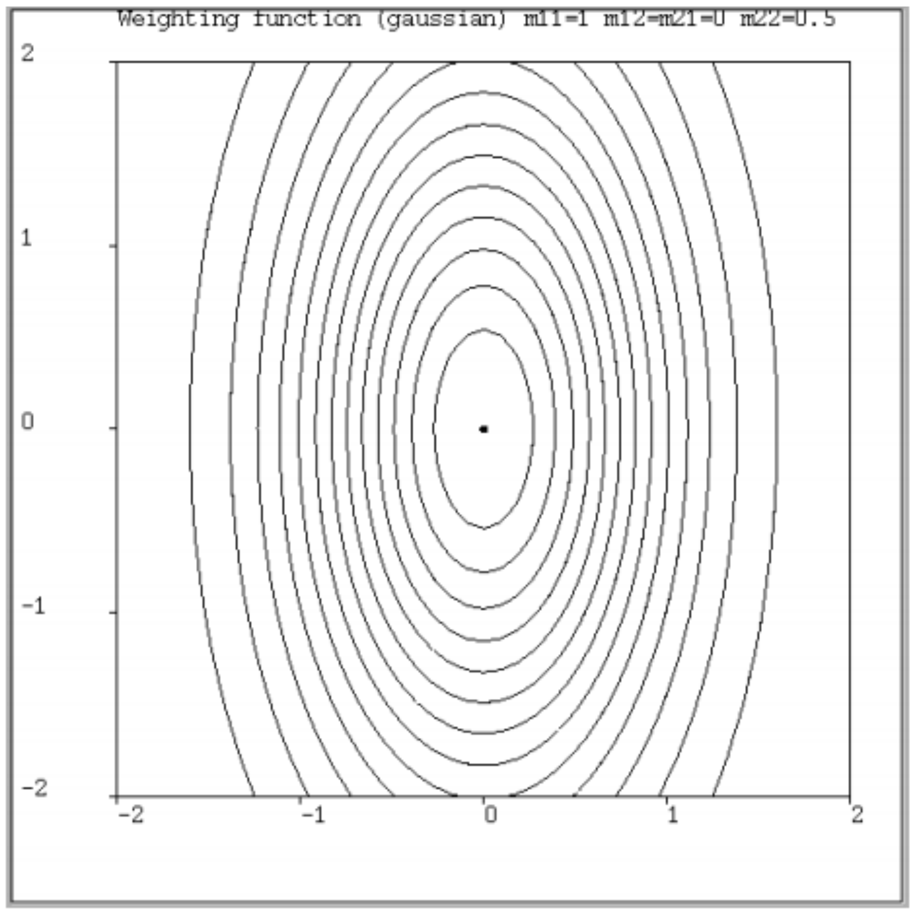
- Use cross-validation to find k
- k should be odd for classification tasks
- General rule of thumb: k < sqrt(N)

k-NN: Normalization of Features

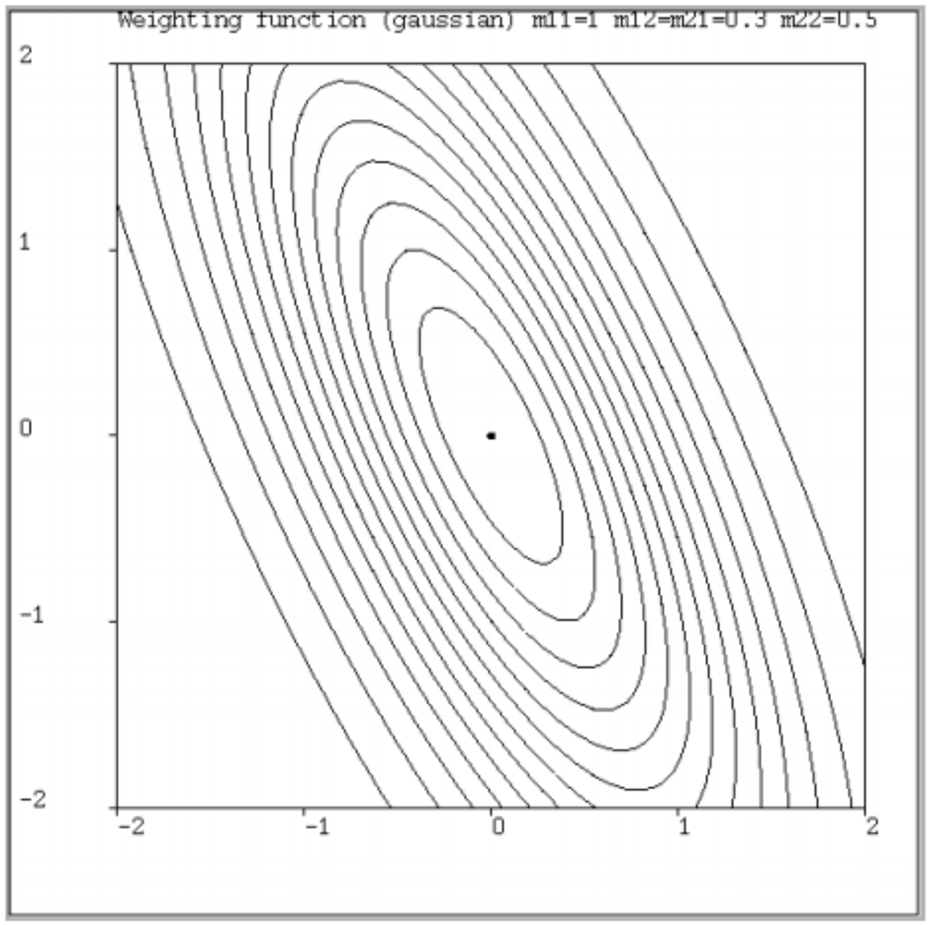
- What if some attributes have larger ranges?
    - Scale: Divide each feature by feature's standard deviation
    - Normalize: Linear scale each dimension to have zero mean and variance of 1
- If we have relative importance of each variable, we can weigh them

$$d_w(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (\sum_d w_d (x_d^{(i)} - x_d^{(j)})^2)^{1/2}$$

k-NN: Notable Distance Metrics



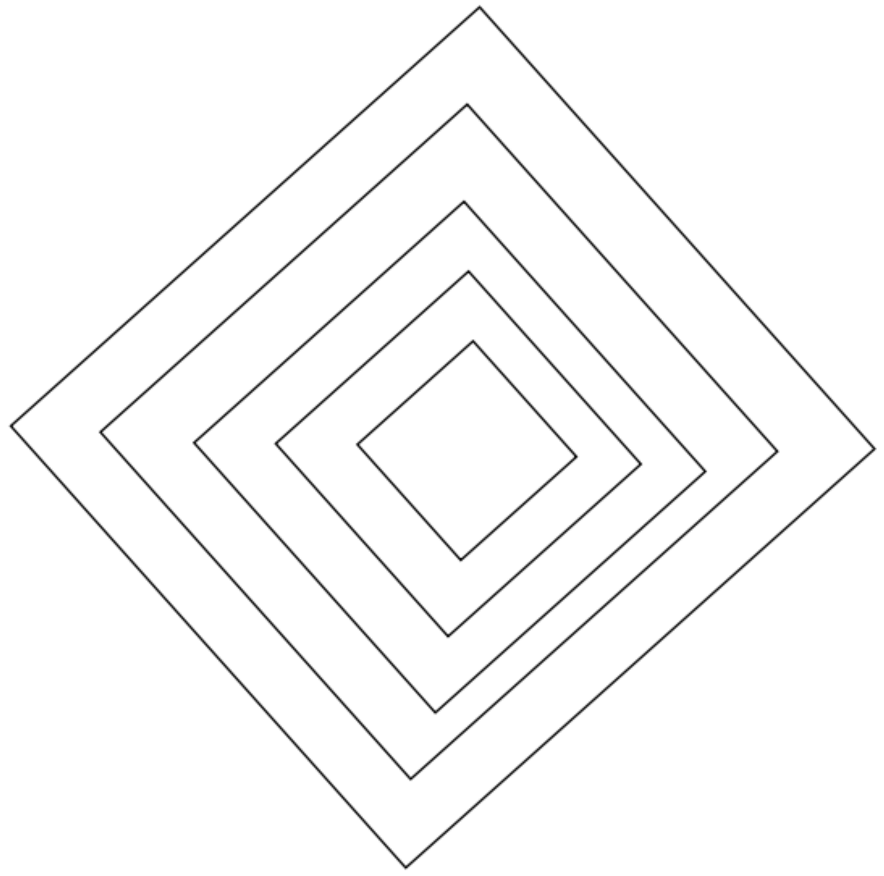
Scaled Euclidean
(diagonal covariance)

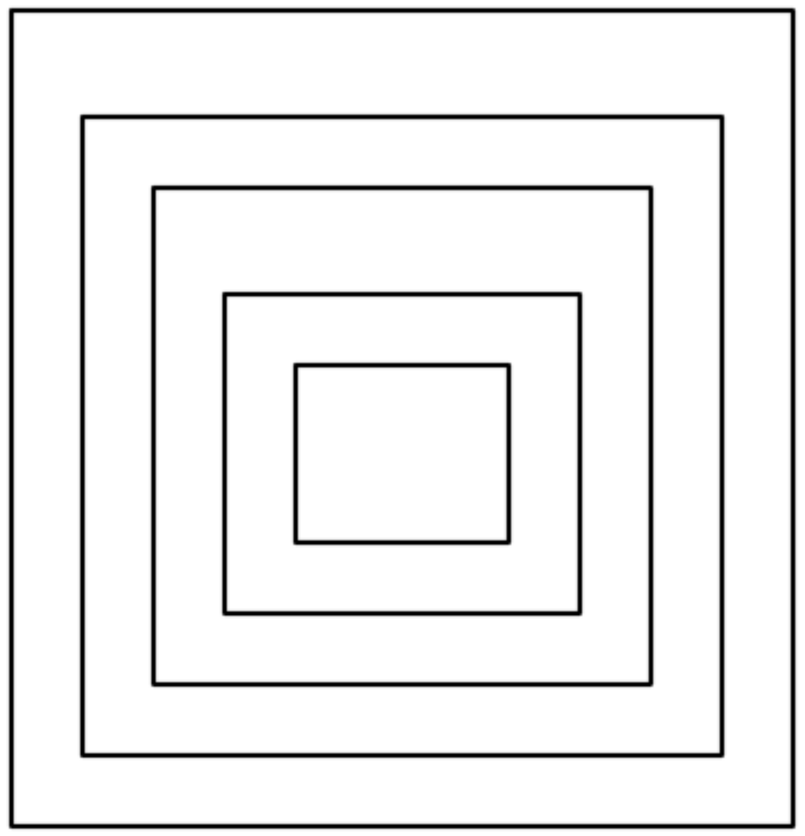


Mahalanobis
(full covariance)

$$D(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^\top \mathbf{S}^{-1} (\mathbf{x} - \mathbf{y})}$$

k-NN: Notable Distance Metrics





Manhattan or taxicab
L_1 norm

$$D(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{p} |x_i - y_i|$$

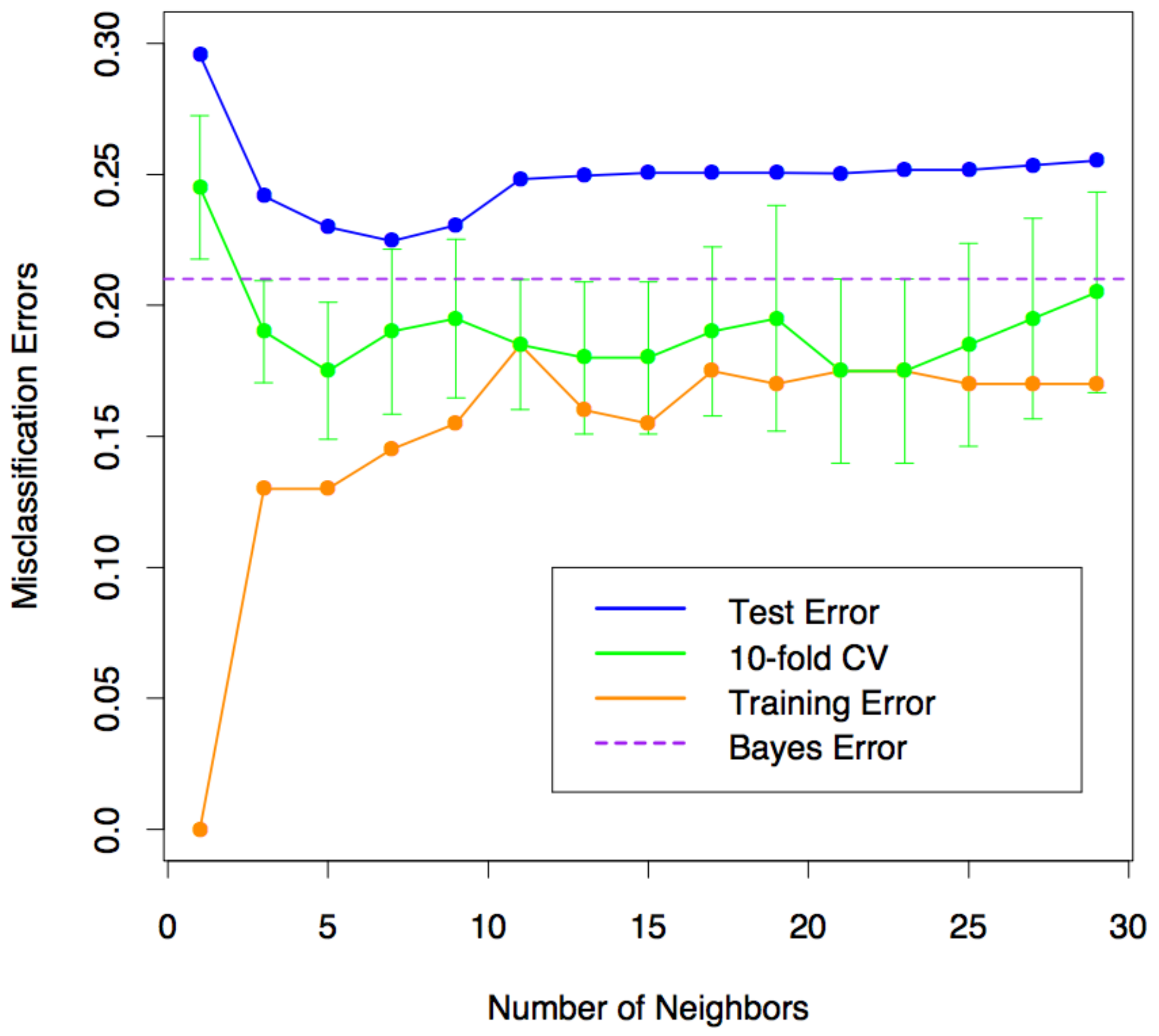
Maximum norm

$$D(\mathbf{x}, \mathbf{y}) = \max |x_i - y_i|$$

k-NN: Notable Distance Metrics

- Integer-valued space
  - Hamming distance: $D(\mathbf{x}, \mathbf{y}) = \frac{N_{\text{different}}(\mathbf{x}, \mathbf{y})}{N_{\text{total}}}$
  - Canberra: $D(\mathbf{x}, \mathbf{y}) = \sum \frac{|x_i - y_i|}{|x_i| + |y_i|}$
- Boolean-valued space
  - Jaccard: $D(\mathbf{x}, \mathbf{y}) = \frac{|\mathbf{x} \cap \mathbf{y}|}{|\mathbf{x} \cup \mathbf{y}|}$
  - Matching: $D(\mathbf{x}, \mathbf{y}) = \frac{N_{\text{same}}(\mathbf{x}, \mathbf{y})}{N_{\text{total}}}$

Example: k-NN Results



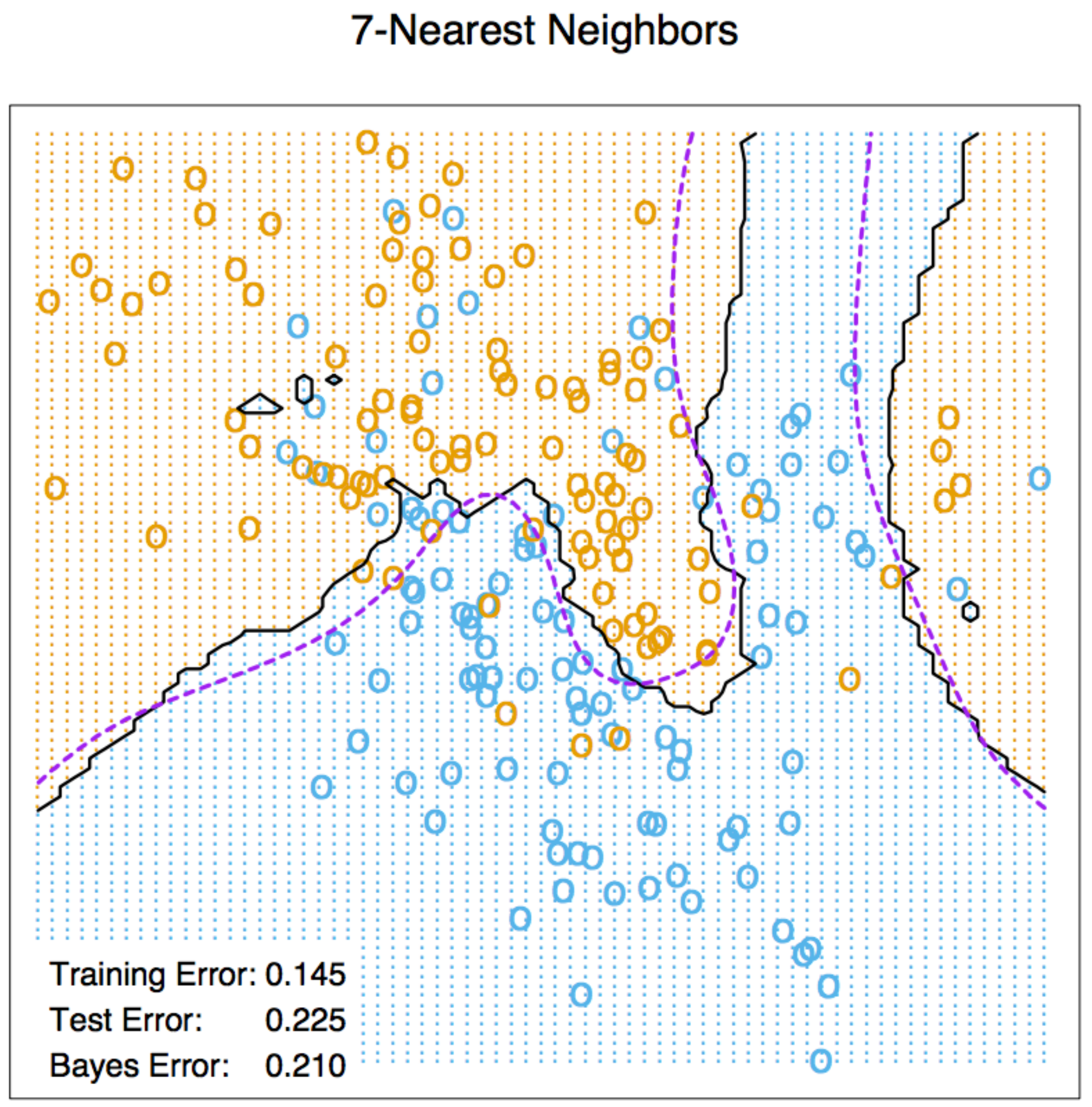


Figure 13.4 (Hastie et al.)

Example: STATLOG Project

- LANDSAT image for classification
- Each pixel has class label from 7-element set
- Classify land usage at pixel based on information from four spectral bands

Elements of Statistical Learning (2nd Ed.) ©Hastie, Tibshirani & Friedman 2009 Chap 13

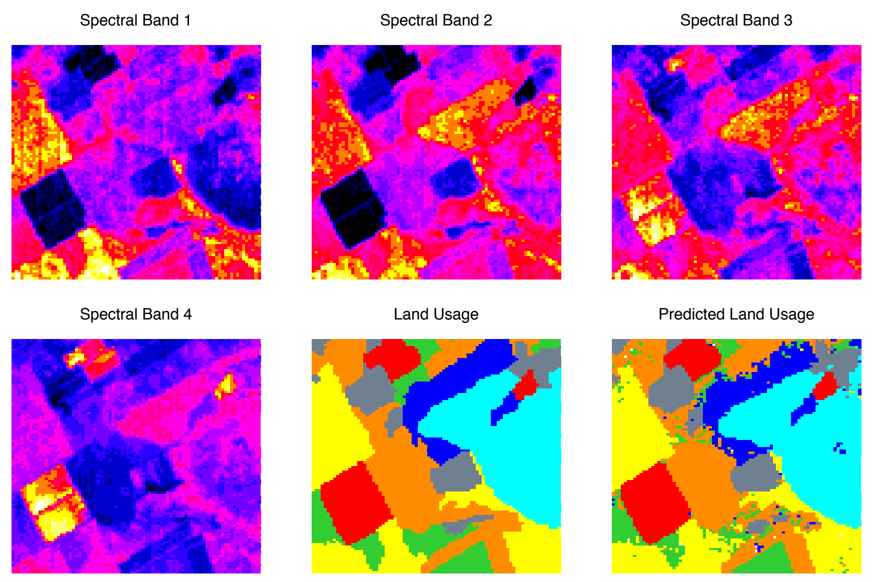


FIGURE 13.6. *The first four panels are LANDSAT images for an agricultural area in four spectral bands, depicted by heatmap shading. The remaining two panels give the actual land usage (color coded) and the predicted land usage using a five-nearest-neighbor rule described in the text.*

Figure 13.6 (Hastie et al.)

Example: STATLOG Project

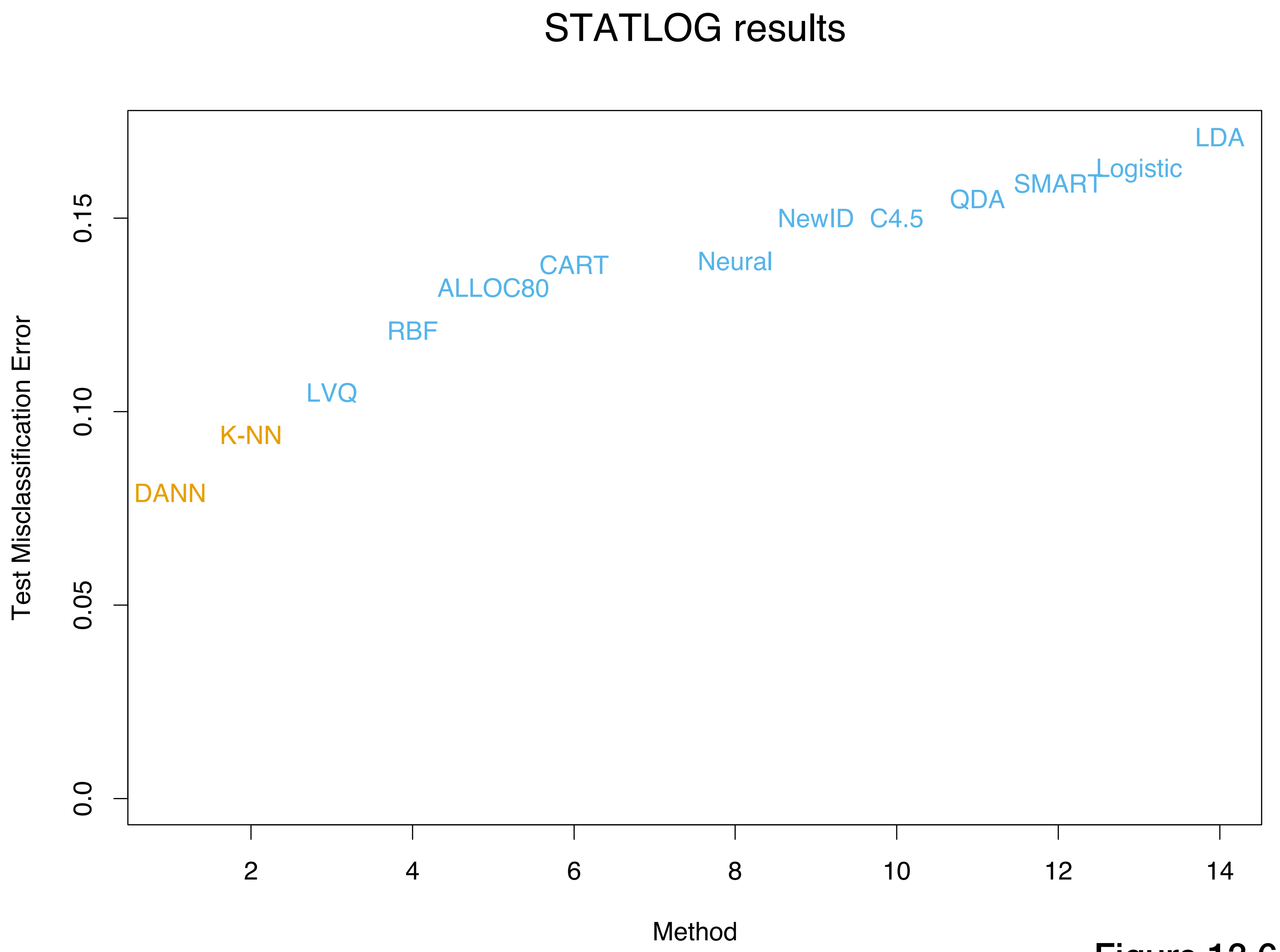
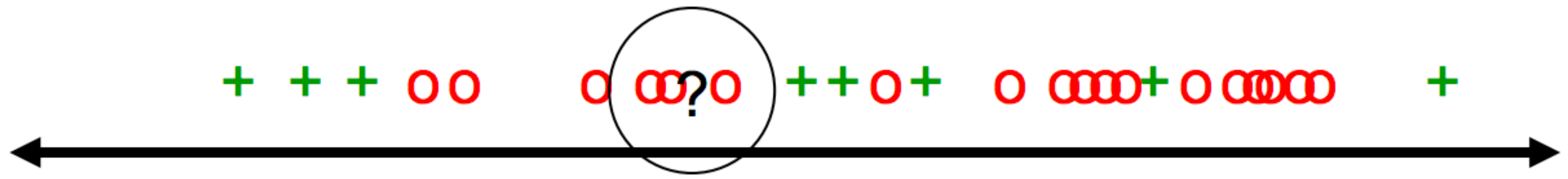


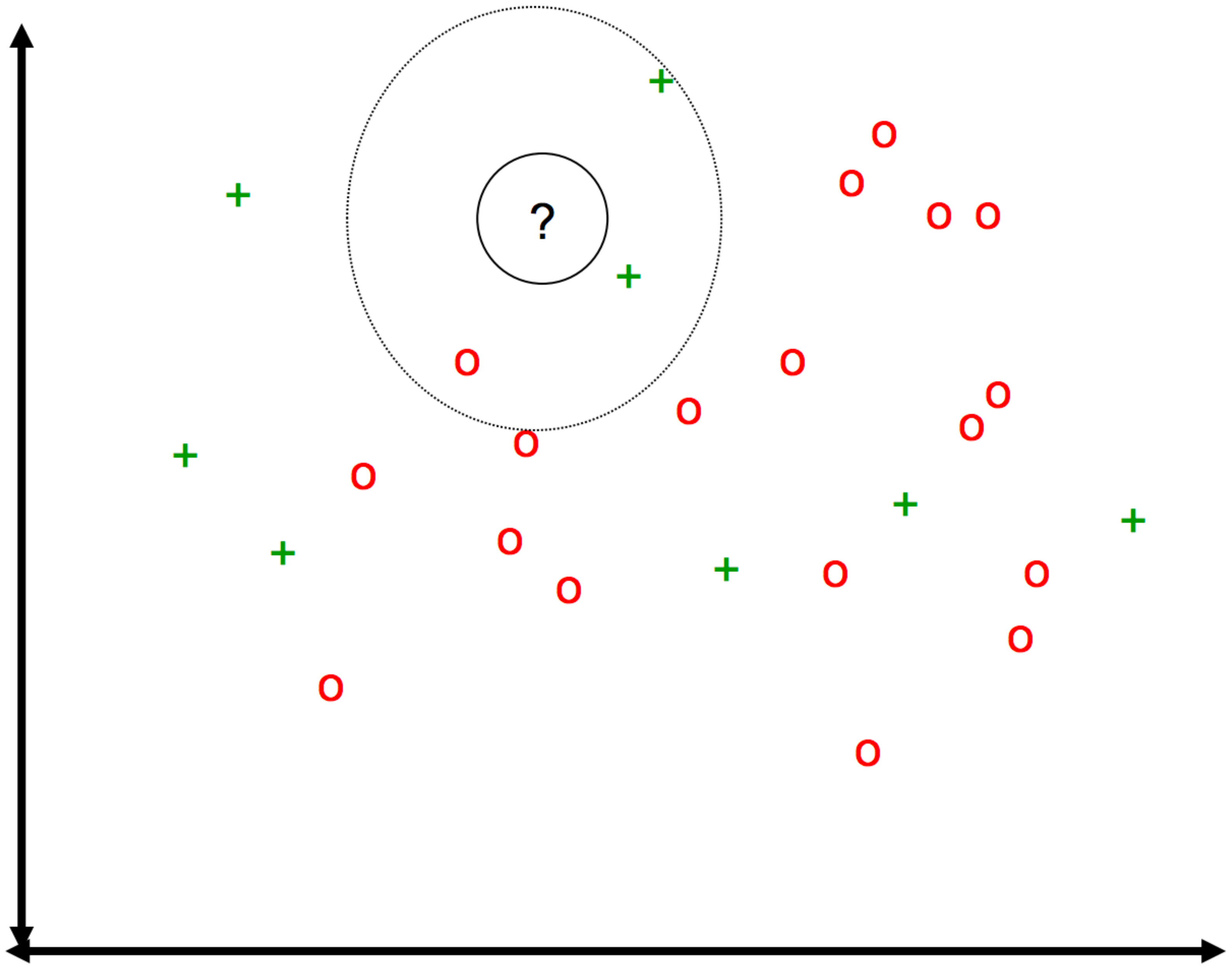
Figure 13.6 (Hastie et al.)

k-NN: Irrelevant Features

- Irrelevant / noisy features add random perturbations to the distance measure
- Hurts performance
- Example: 1-D data, what happens if we add noisy attribute?



k-NN: Irrelevant Features



k-NN: Curse of Dimensionality

- Consider N data points uniformly distributed in unit cube
- Unit cube of size $\left[-\frac{1}{2}, \frac{1}{2}\right]^p$
- Let R be radius of 1-NN centered at origin

$$\text{median}(R) = v_p^{-1/p} \left(1 - \frac{1}{2}^{1/N}\right)^{1/p}, v_p(r) = \frac{2r^p \pi^{p/2}}{p\Gamma(p/2)}$$

Example: Curse of Dimensionality

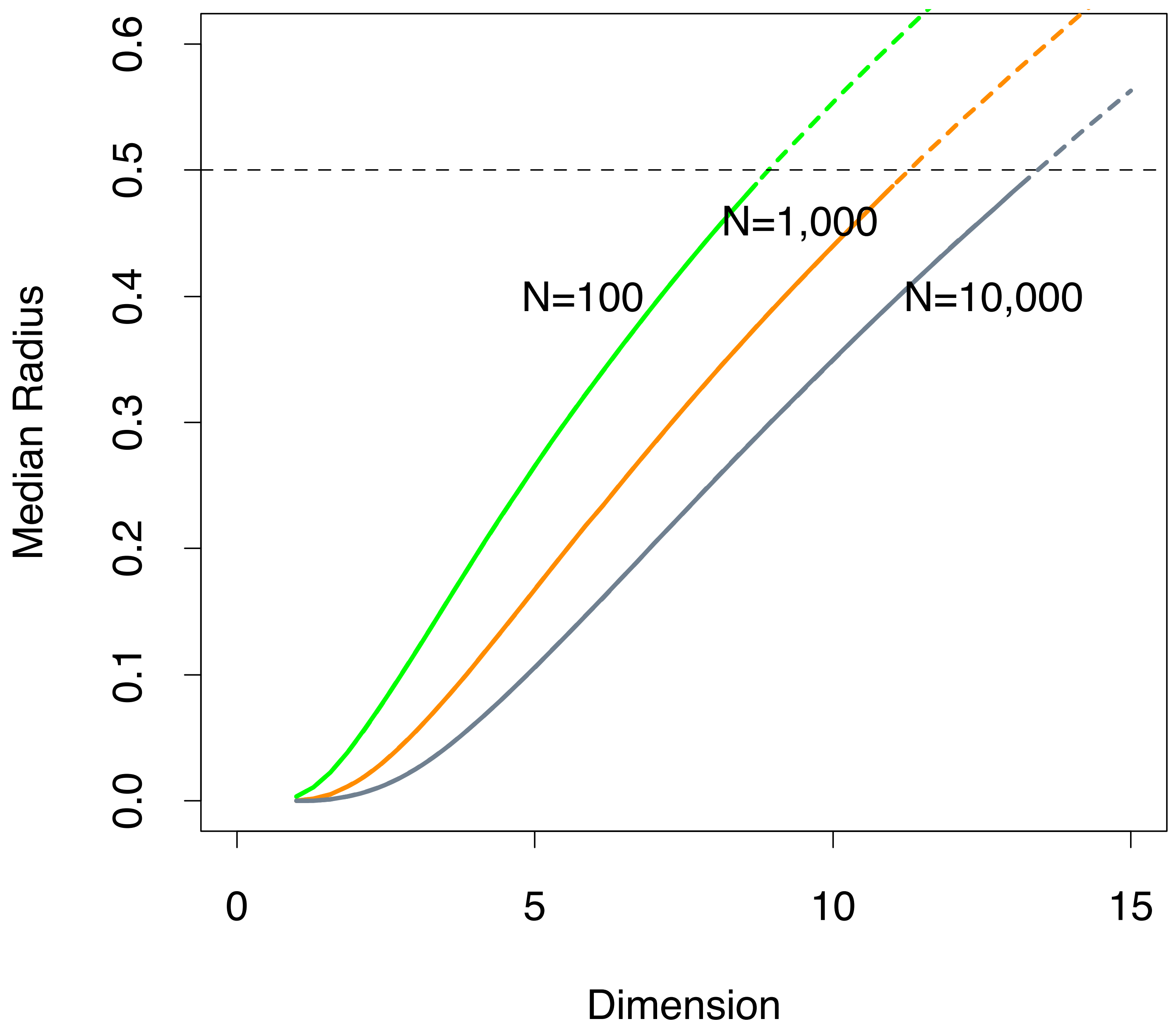


Figure 13.12 (Hastie et al.)

k-NN: Curse of Dimensionality II

- Suppose we have 1000 points uniformly distributed on unit hypercube with a query point at the origin
- Apply 5-nearest neighbor
  - 1-D: $(5/1000) = 0.005$
  - 2-D: $\sqrt{5/1000} \approx 0.0707$
  - 30-D: $(5/1000)^{1/30} \approx 0.8381$

Distances concentrate within a small range and all points look “equidistant”!

k-NN: Efficient Indexing

- Linear algorithm (no pre-processing)
    - Query: $\Omega(Np)$
- Voronoi diagram
    - Query: $O(\log N)$
    - Memory: $O\left(p^{\lceil N/2 \rceil}\right)$
- Tree-based data structures (pre-processing)

kd-Trees

kd-Trees [Bentley ’75]

- Binary tree (data structure) for storing finite set of points from a k-dimensional space
- Idea: Each level of the tree compares against 1 dimension
- Not the most efficient solution in theory but used in practice
- Name originally meant 3d-trees, 4d-trees, …, where k was the number of dimensions

kd-Trees

- Applications
    - Nearest neighbor search
    - Range queries
    - Fast look-up
- Guaranteed $\log_2 n$ depth where n is the number of points in the set

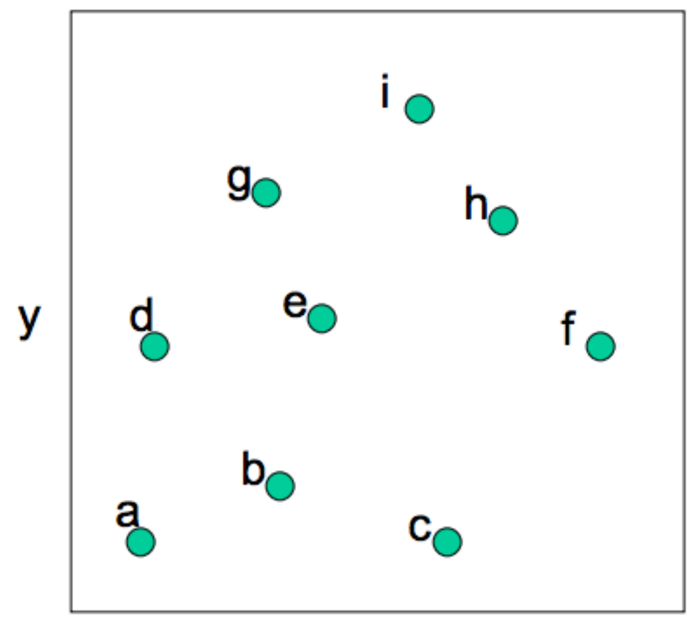
kd-Tree: Construction

- Binary tree with:
    - Size: O(n)
    - Depth: O(log n)
    - Construction: O(n log n)

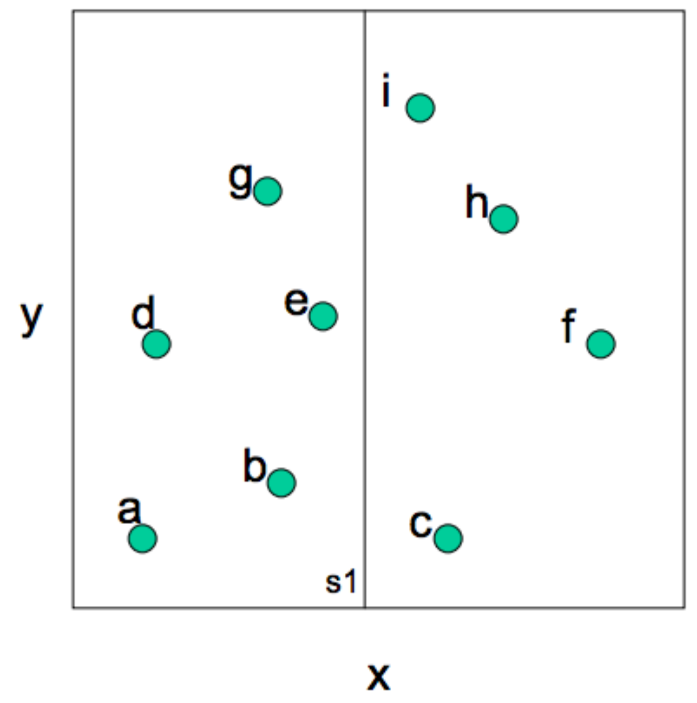
kd-Tree: Construction

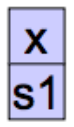
- If just one point, form a leaf with that point
- Otherwise, choose an axis and divide the points in half via the median of the axis
- Recursively construct kd-trees for the two sets of points

Example: kd-Tree Construction



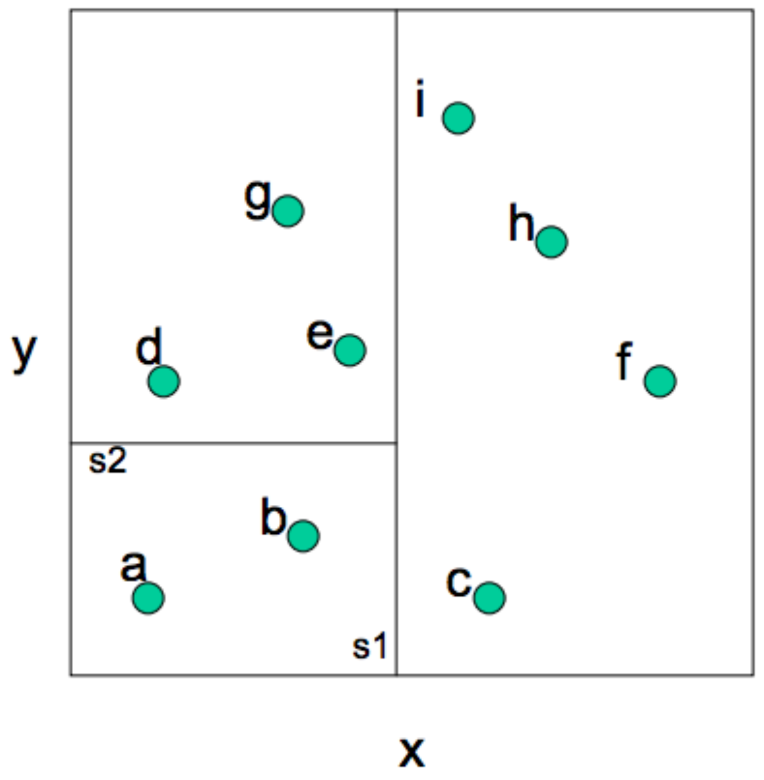
Select an axis (x) and choose the median line

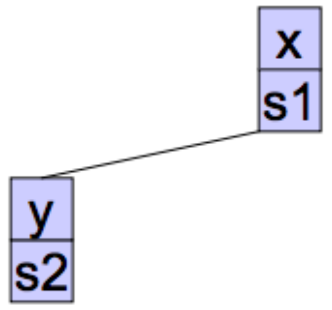


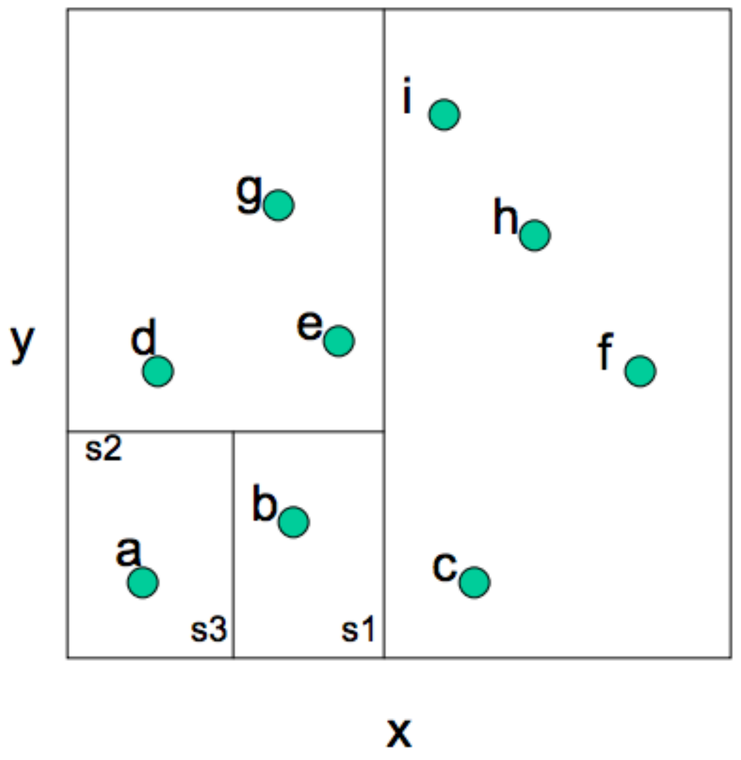


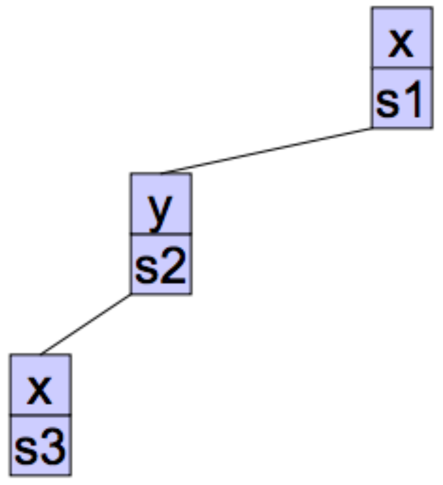
https://courses.cs.washington.edu/courses/cse373/02au/lectures/lecture22l.pdf

Example: kd-Tree Construction

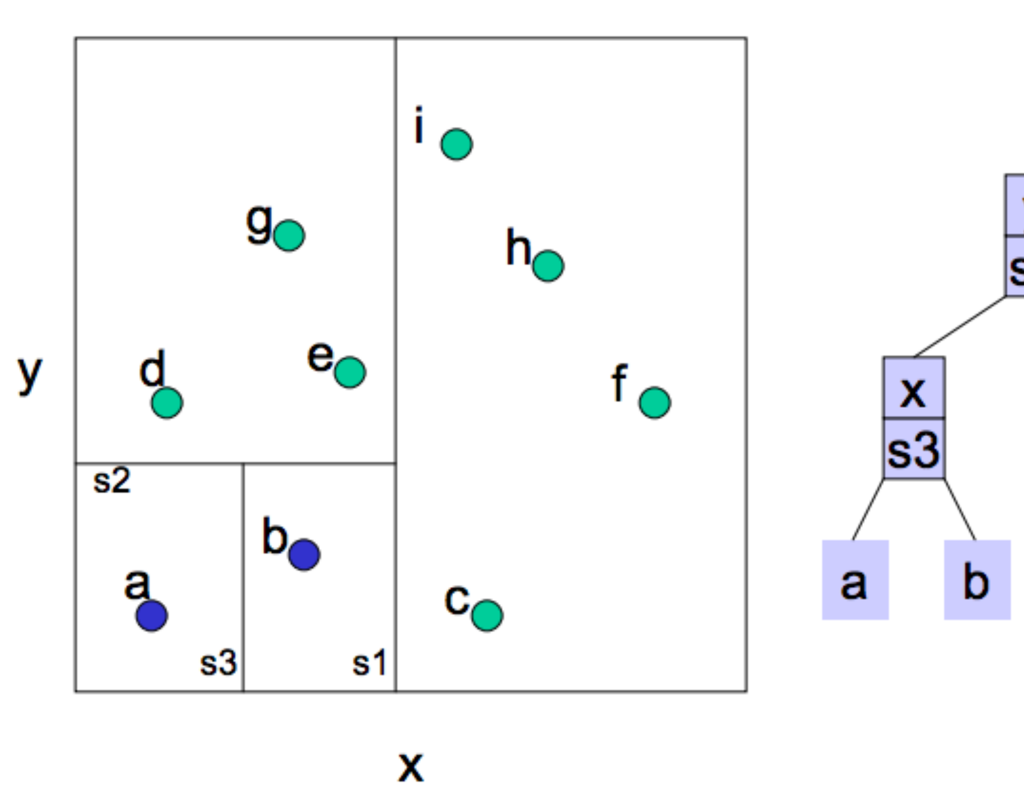


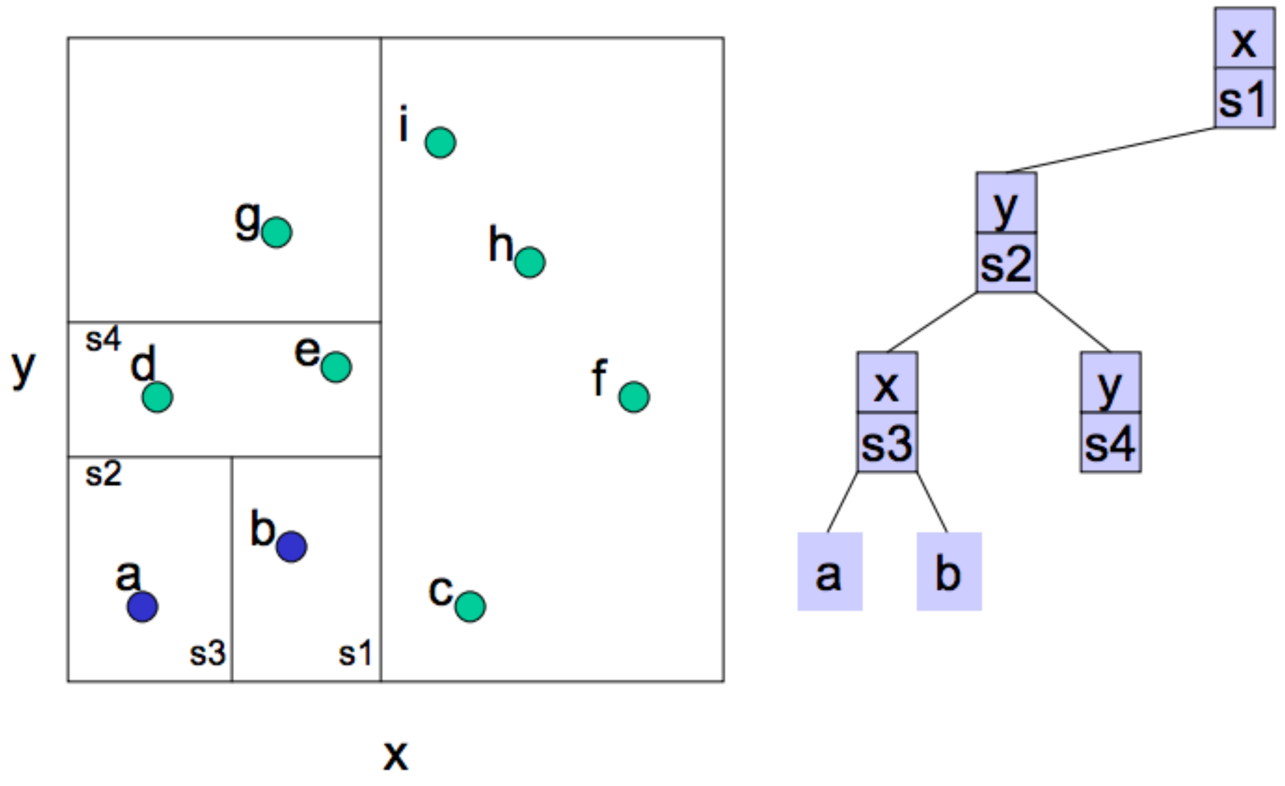




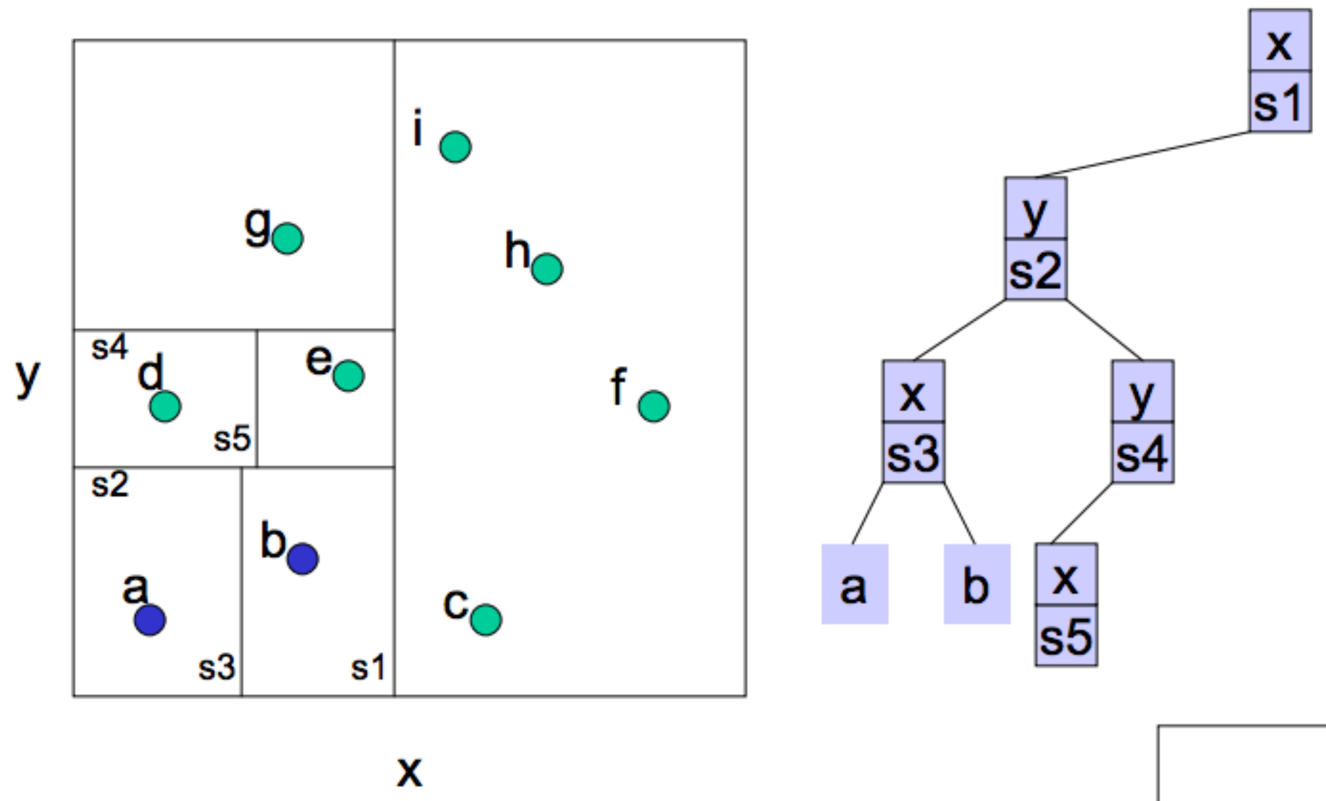


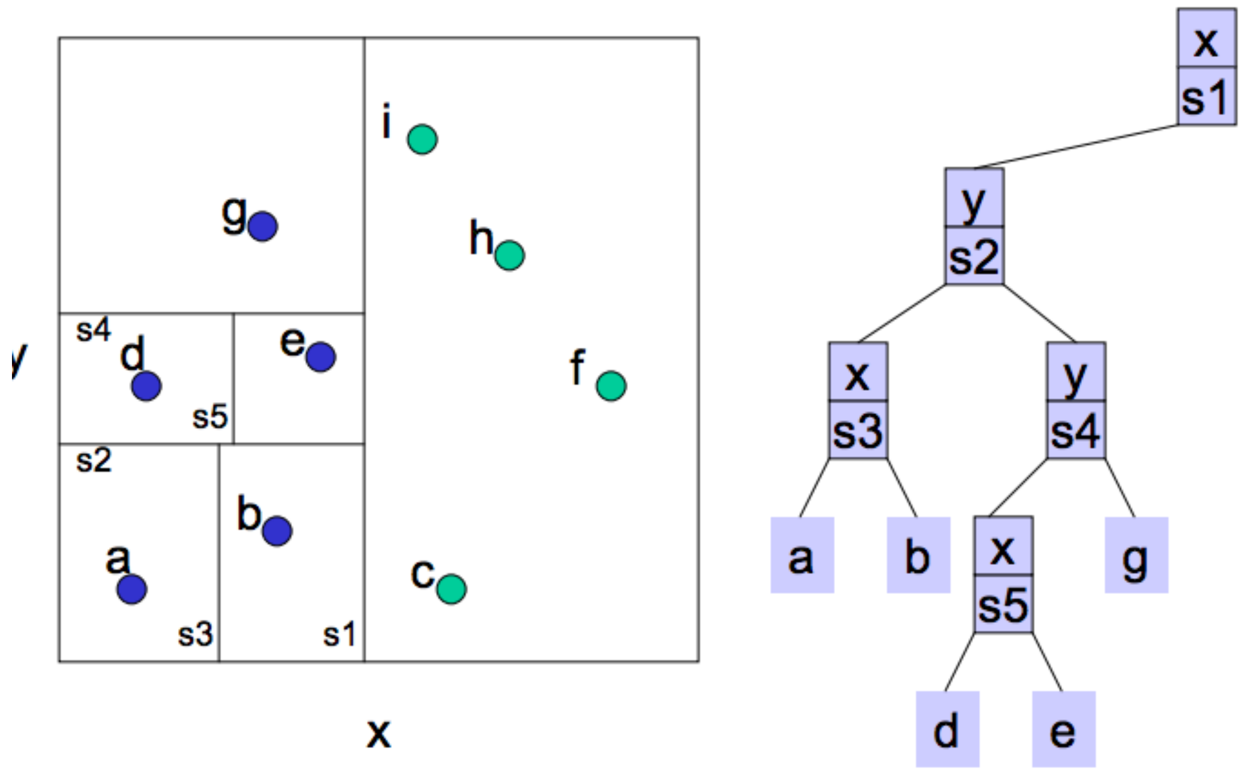
Example: kd-Tree Construction



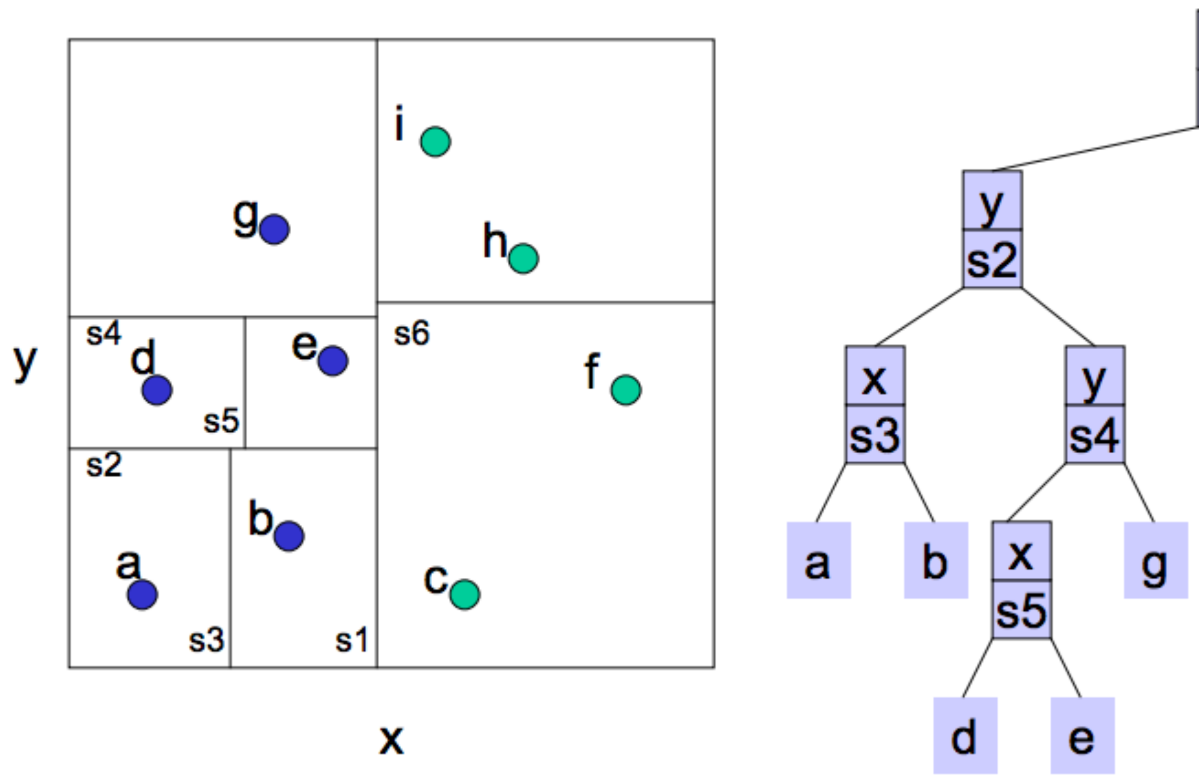


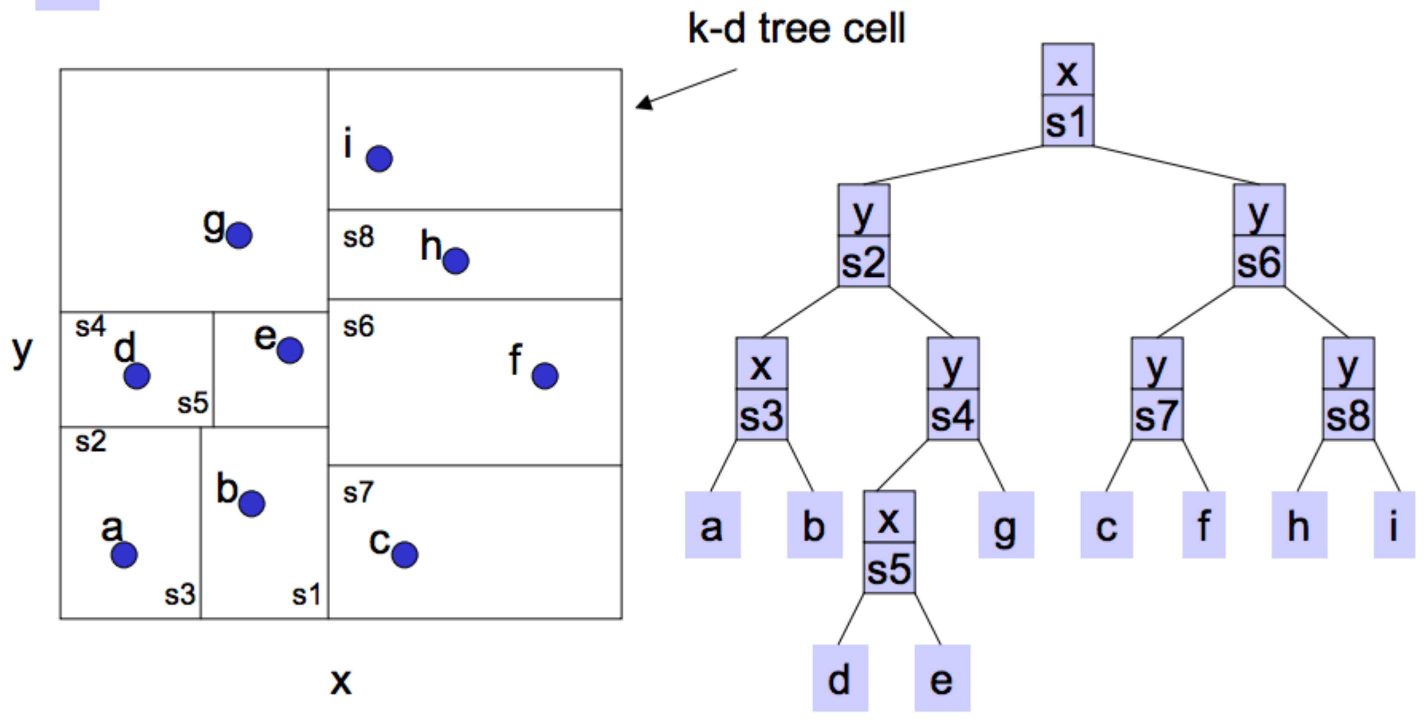
Example: kd-Tree Construction





Example: kd-Tree Construction

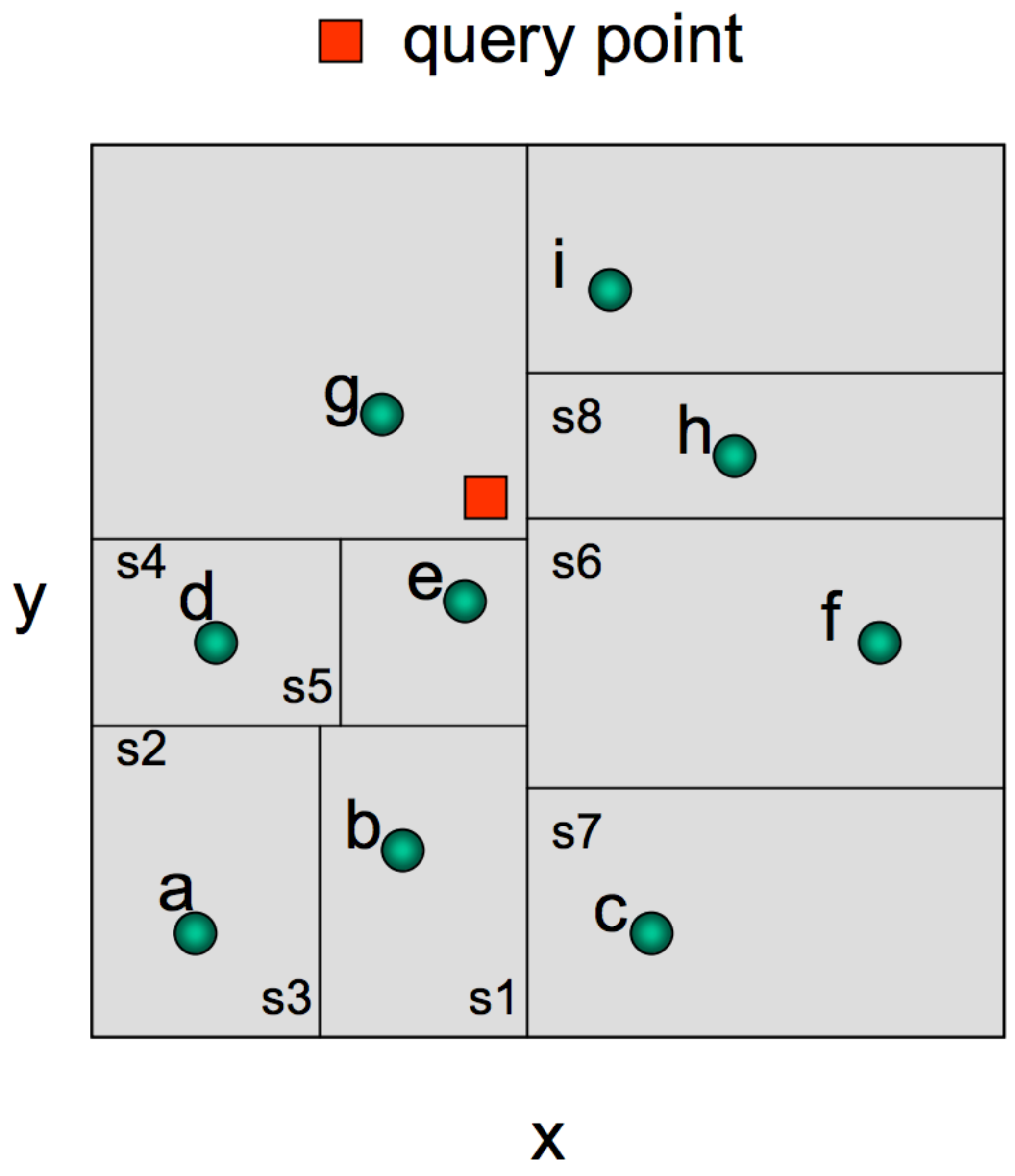


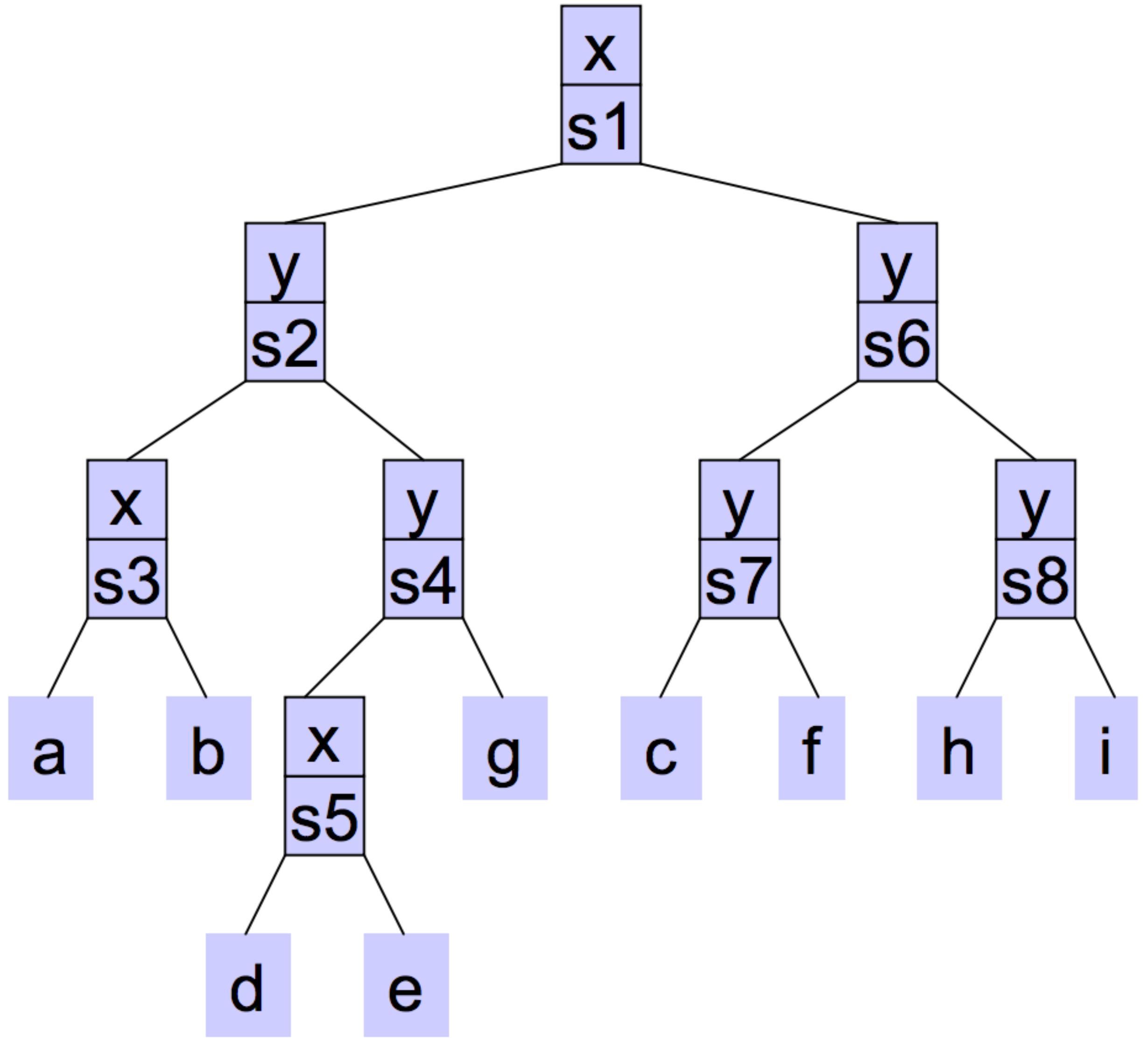


kd-Tree: NN Search

- Search recursively to find the point in the same cell as the query
- On return search each subtree where a closer point other than the one you know about might be found
- Has been shown to run in O(log n) average time per search in a reasonable model (assuming p is constant)

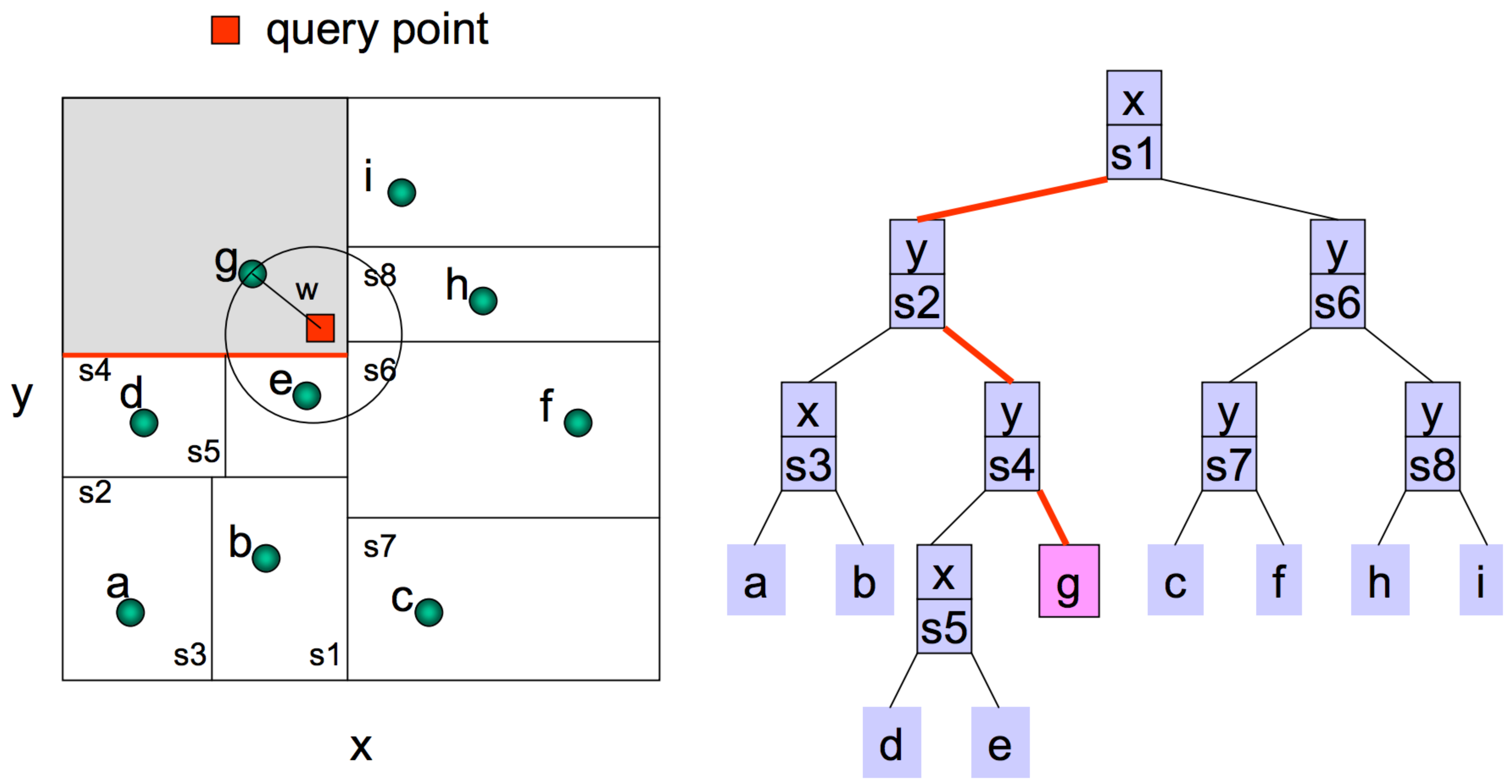
Example: kd-Tree Query



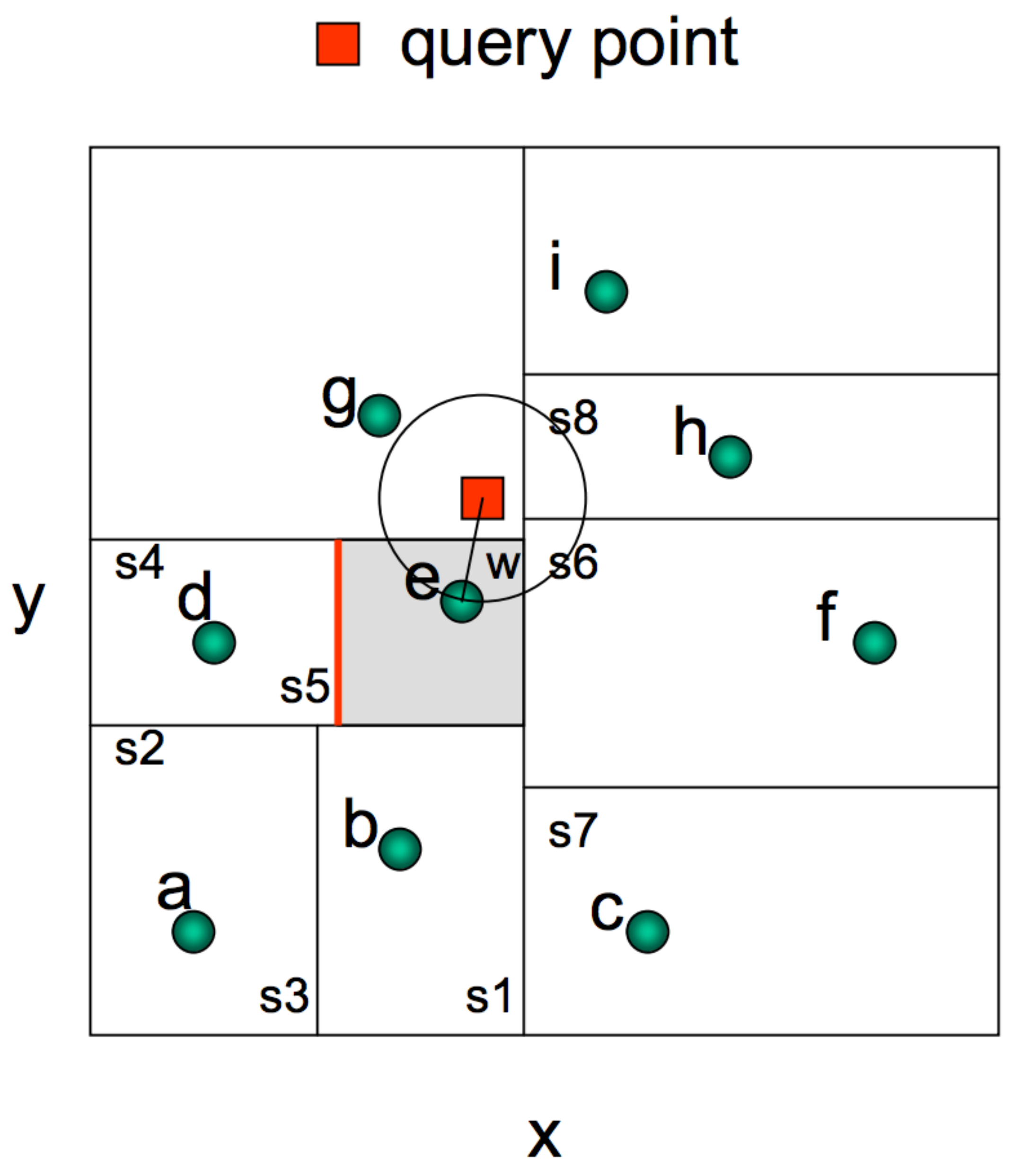


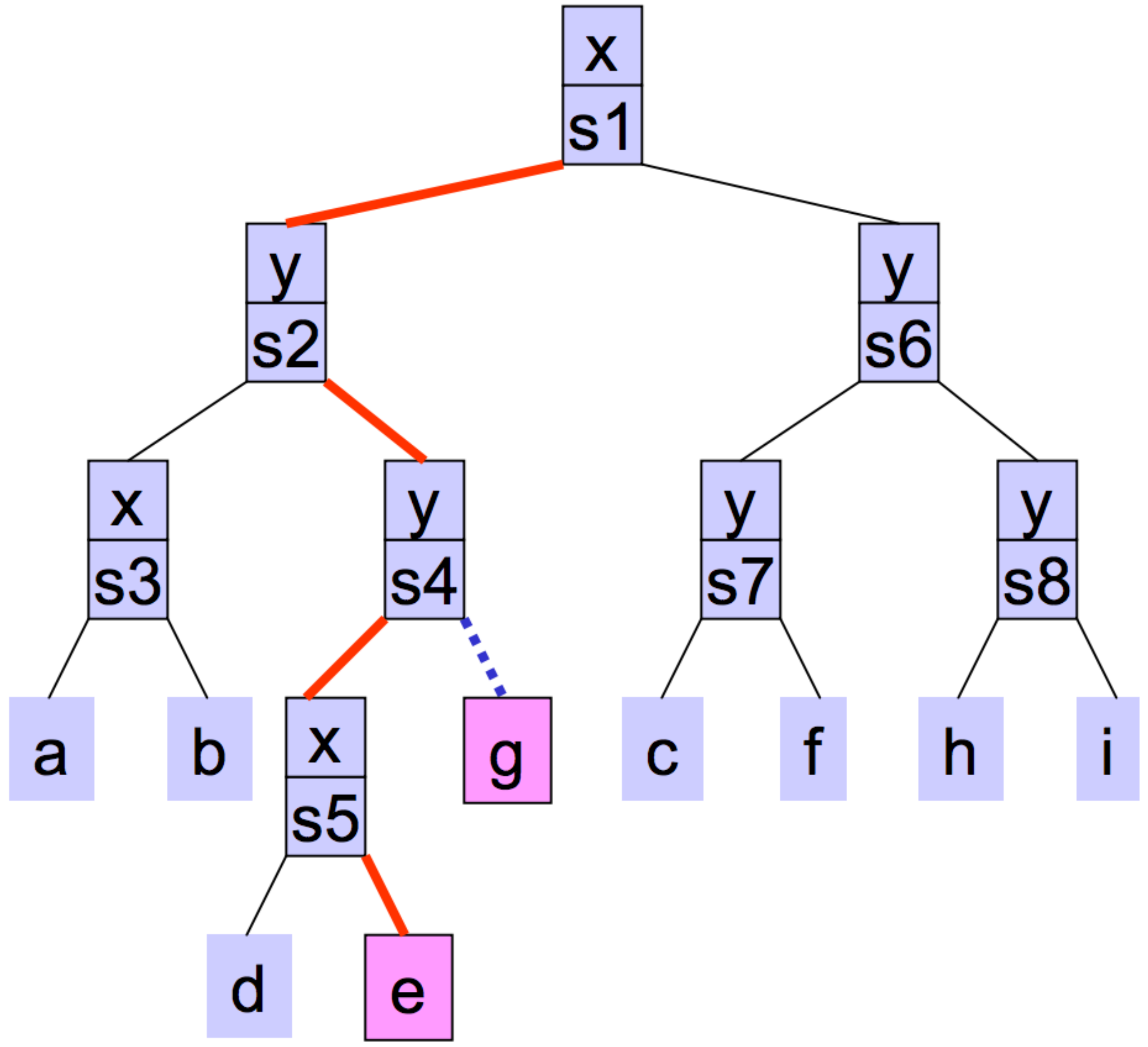
https://courses.cs.washington.edu/courses/cse373/02au/lectures/lecture22l.pdf

Example: kd-Tree Query

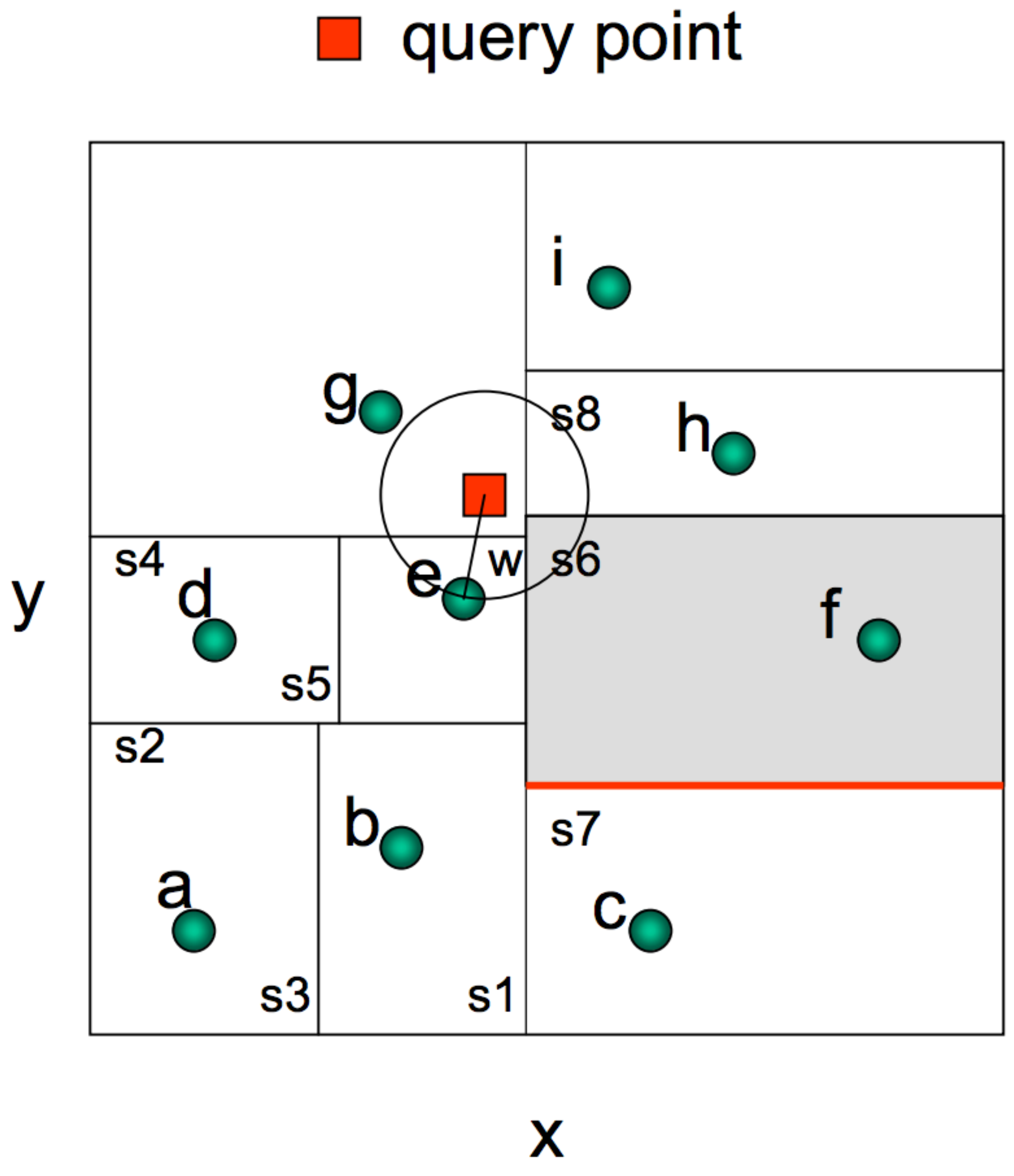


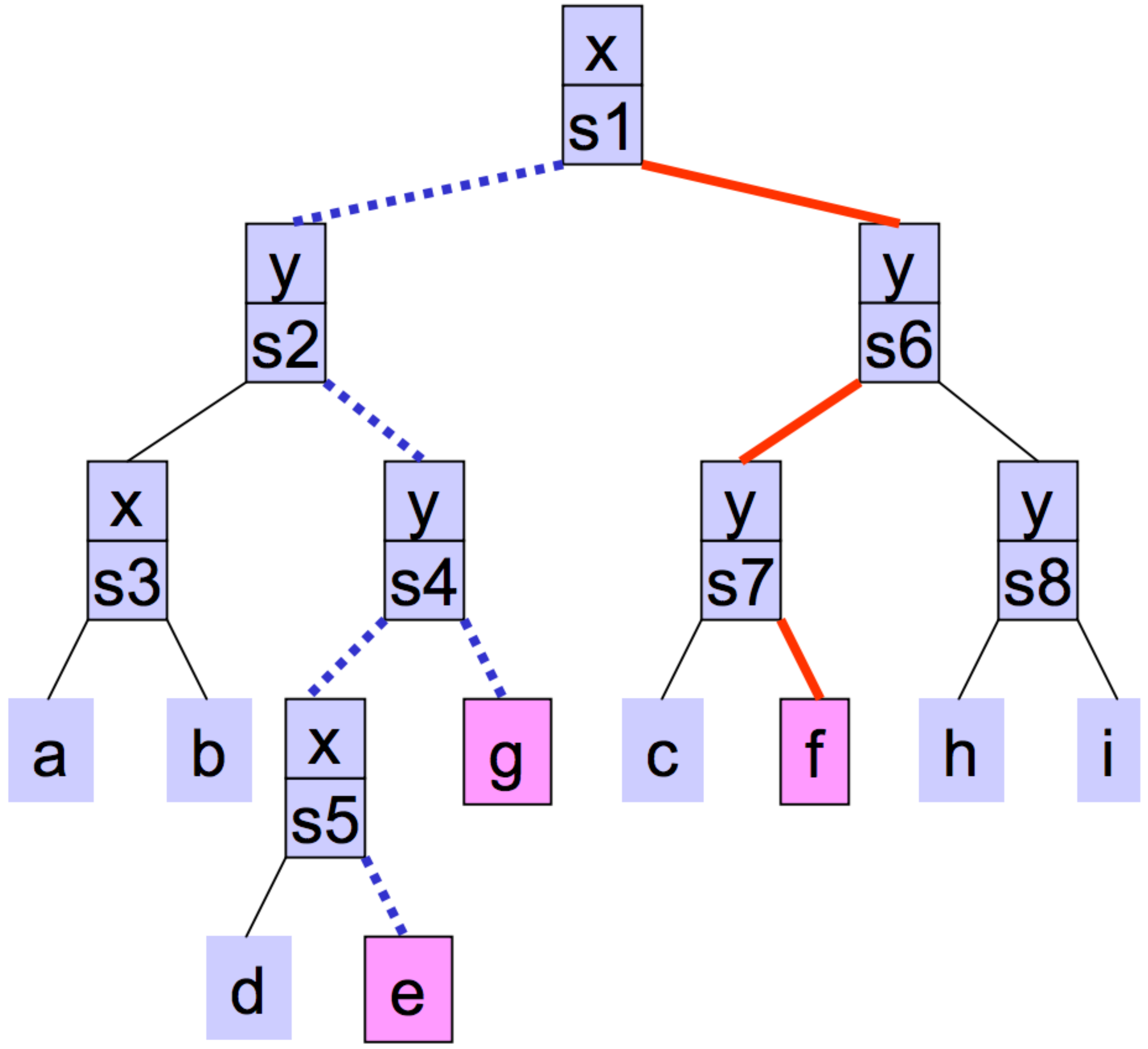
Example: kd-Tree Query



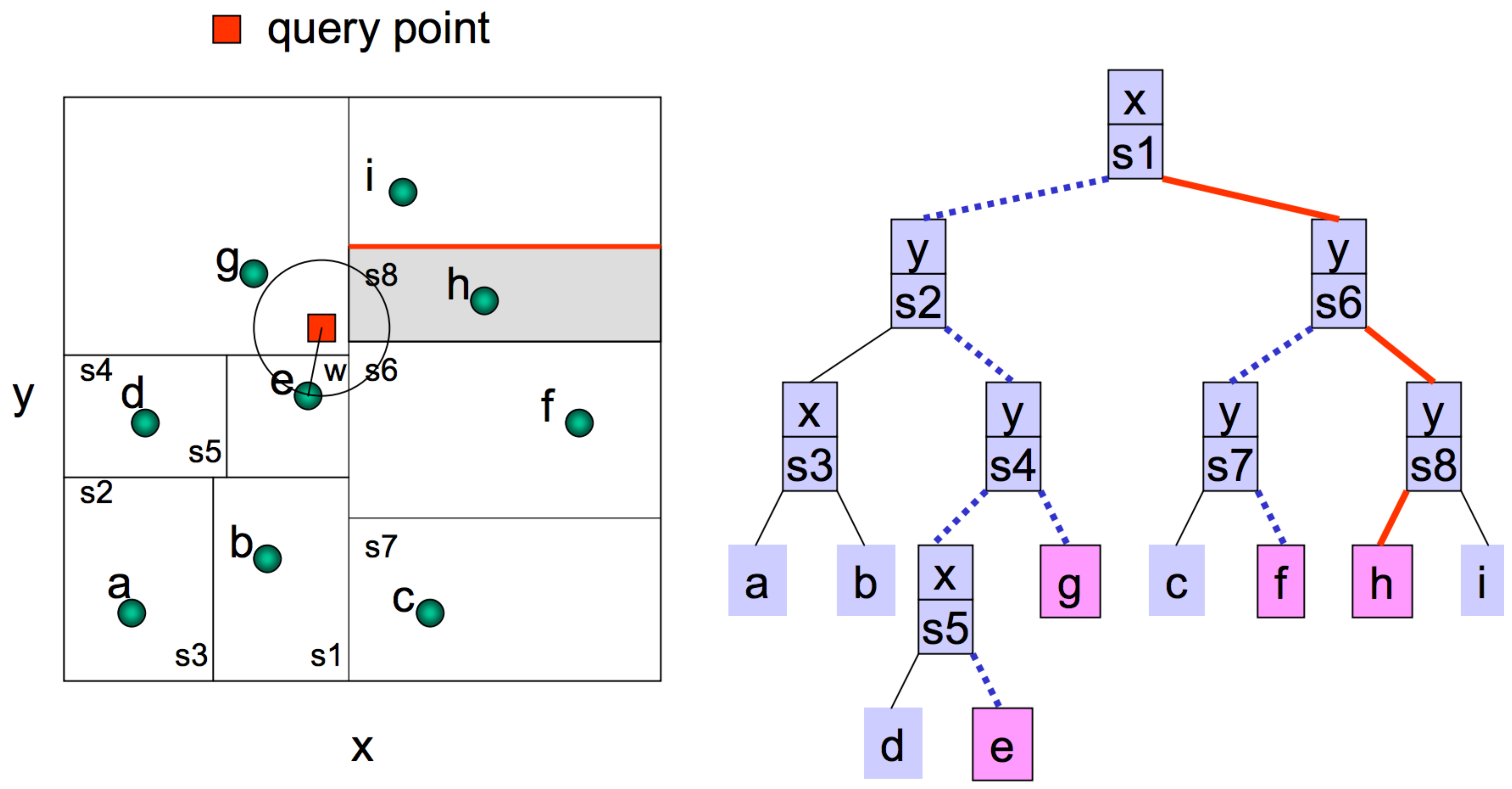


Example: kd-Tree Query





Example: kd-Tree Query

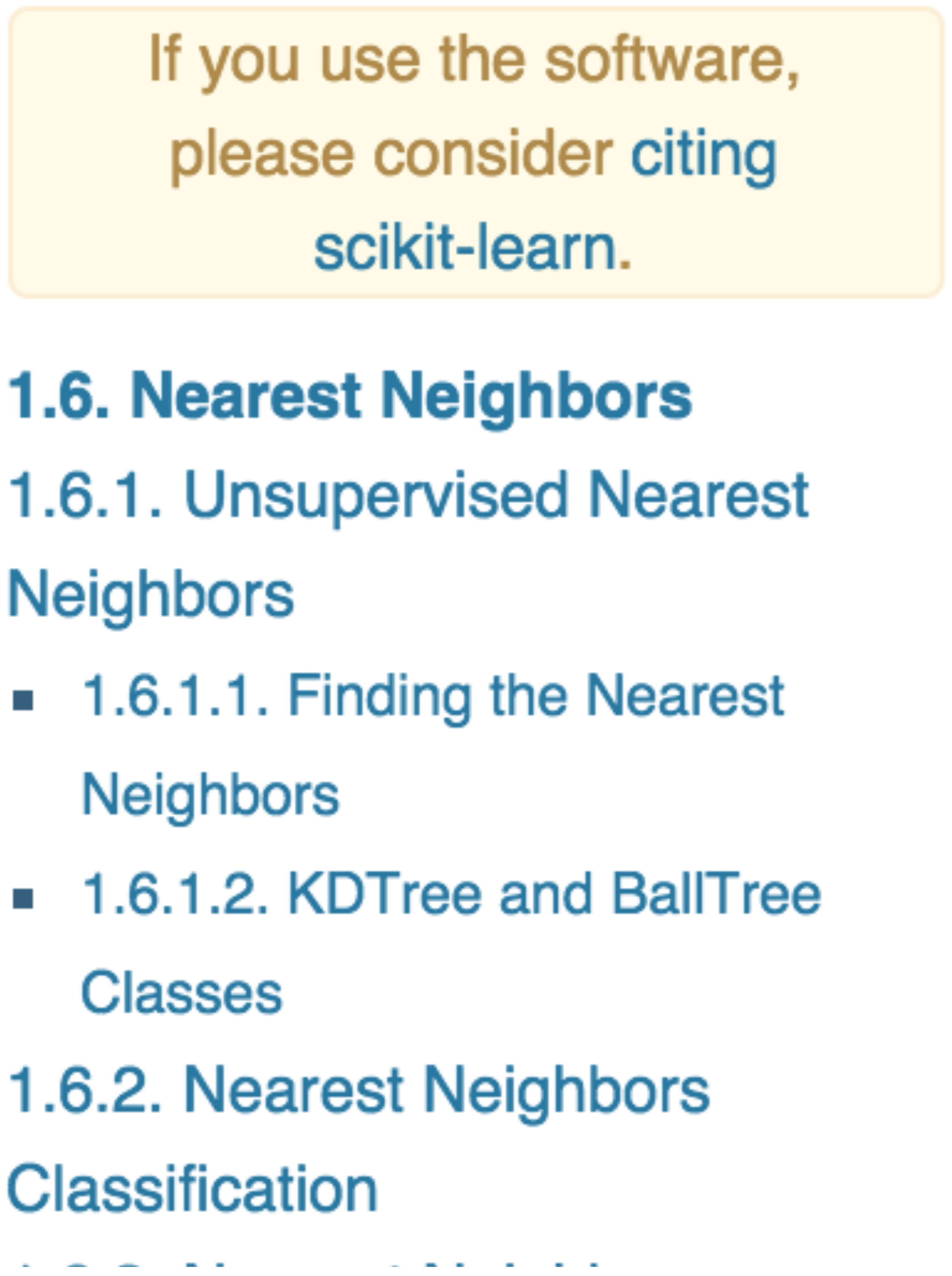


kd-Tree Summary

- Tons of variants
    - Construction of trees (e.g., heuristics for splitting, stopping, representing branches)
    - Other representational data structures for fast NN search (e.g., ball trees, …)
- High-dimensional spaces are hard

Python: Scikit-learn

- Brute force: $O(pn^2)$
- kd-tree: $O(p \log n)$ for $p < 20$
- Ball tree: $O(p \log n)$ but tree construction is more costly than kd-tree
- Benchmarking using NN: https://jakevdp.github.io/blog/2013/04/29/benchmarking-nearest-neighbor-searches-in-python/



If you use the software, please consider citing scikit-learn.

1.6. Nearest Neighbors

1.6.1. Unsupervised Nearest Neighbors

- 1.6.1.1. Finding the Nearest Neighbors
- 1.6.1.2. KDTree and BallTree Classes

1.6.2. Nearest Neighbors Classification

Weighted k-NN

- Generalization of k-NN: weigh the i^th training point by how far **x**$_i$ is from **x**

$$\hat{y}(\mathbf{x}) = \text{sign}\left(\sum_{i}^{N} y_i D(\mathbf{x}_i, \mathbf{x})\right)$$

- Possible weight functions

$$D(\mathbf{x}_i, \mathbf{x}) = \frac{1}{||\mathbf{x}_i - \mathbf{x}||_2^2}$$

$$D(\mathbf{x}_i, \mathbf{x}) = \exp\left(-\gamma ||\mathbf{x}_i - \mathbf{x}||_2^2\right)$$

k-NN: Similarity to kernel SVM

- Recall Gaussian kernel SVM

$$\hat{y}(\mathbf{x}) = \text{sign}\left(\sum_{i}^{N} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x})\right),$$

$$K(\mathbf{u}, \mathbf{v}) = \exp(-\gamma ||\mathbf{u} - \mathbf{v}||_2^2)$$

- Discriminant functions are nearly identical — SVM just learns the parameters

k-NN: Advantages

- Instance-based (lazy) learning
  - Training model is cheap & fast
  - No optimization required
- Easy to understand and implement
- Can be quite accurate — models complex decision boundaries quite well

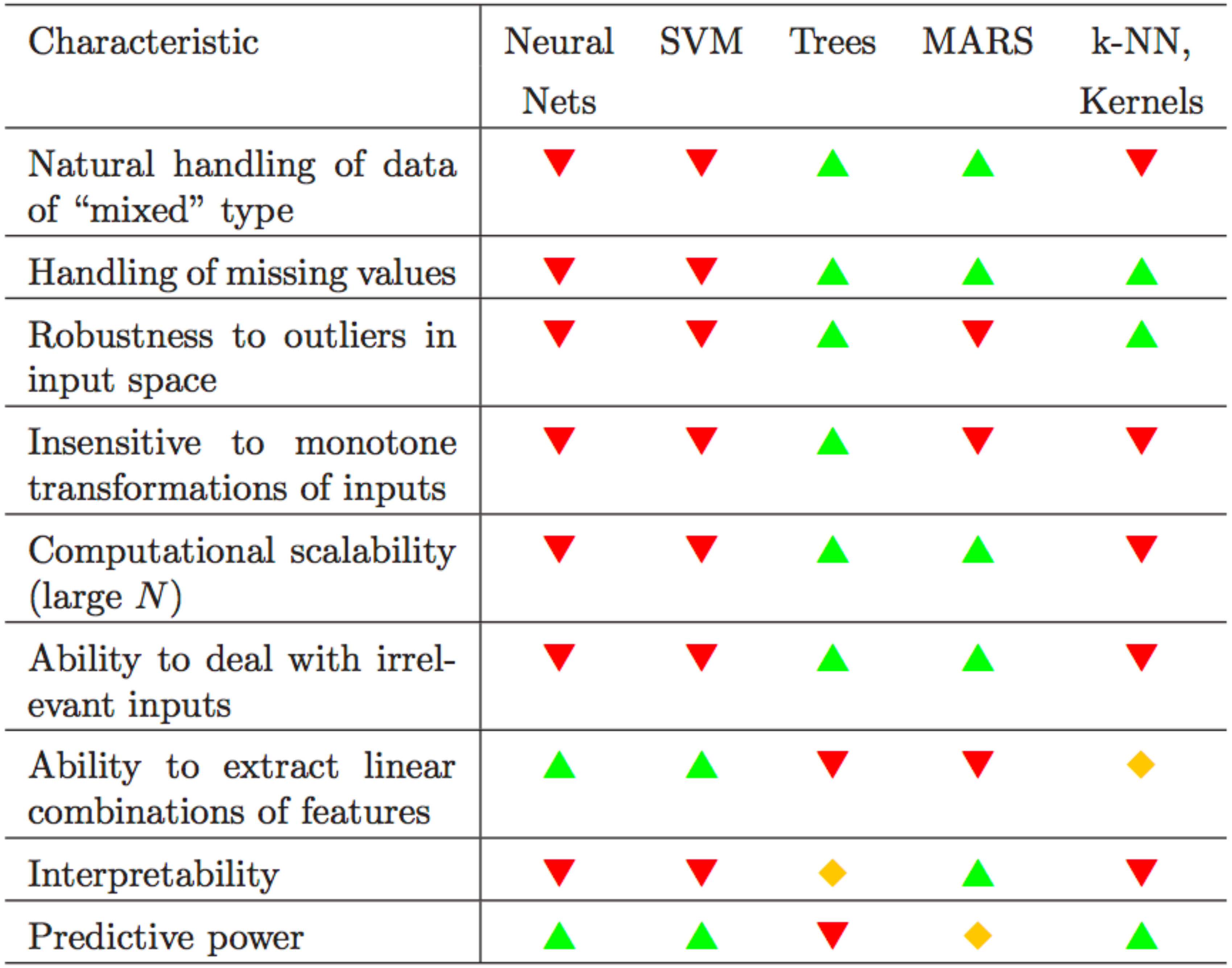
k-NN: Disadvantages

- Slow at query time: O(Np)
- Memory intensive — must store all training examples
- Easily fooled by irrelevant features
- Suffers from curse of dimensionality

k-NN: When to Consider

- Instance map to points in real space
- Less than 20 attributes per instance
- Lots of training data

Model Comparison



Characteristic	Neural Nets	SVM	Trees	MARS	k-NN, Kernels
Natural handling of data of "mixed" type	▼	▼	▲	▲	▼
Handling of missing values	▼	▼	▲	▲	▲
Robustness to outliers in input space	▼	▼	▲	▼	▲
Insensitive to monotone transformations of inputs	▼	▼	▲	▼	▼
Computational scalability (large N)	▼	▼	▲	▲	▼
Ability to deal with irrelevant inputs	▼	▼	▲	▲	▼
Ability to extract linear combinations of features	▲	▲	▼	▼	◆
Interpretability	▼	▼	◆	▲	▼
Predictive power	▲	▲	▼	◆	▲

Table 10.1 (Hastie et al.)

Approximate Nearest Neighbors

Approximate Nearest Neighbor (ANN)

- Idea: rather than retrieve the exact closest neighbor, make a “good guess” of the nearest neighbor
- c-ANN: for any query q and points p:
    - r is the distance to the exact nearest neighbor q
    - Returns p in P, $||p - q|| \leq cr$, with probability at least $1 - \delta, \delta > 0$

ANN: Altering kd-Tree Search

(Augmented) kd-Trees are used but interrupt search earlier [Arya et al., 1994]

- Prune when distance to bounding box is greater than some distance r over some value alpha
- Saves lots of search time by removing some nodes of the tree
- In practice can get O(p log n) but worst case still has exponential running time

Beyond kd-Trees

- Works for low and medium dimensional data, but has problems with high-dimensional data
- Non-trivial to implement efficiently and still requires some computation of object similarities
- Can we represent similarities between objects in a more succinct manner?
    - Sacrifice exactness for efficiency by using randomization
    - Obtain a “sketch” of the object instead

Johnson-Lindenstrauss Lemma

Main Idea: small set of points in high-dimensional space can be embedded into a space of much lower dimension in such a way that distances between the points are nearly preserved

- One proof of the lemma uses projection onto random subspace
- Used in compressed sensing, manifold learning, dimensionality reduction, and graph embedding

Hash Functions

- A hash function, h, is a function which transforms a key from a set K, into an index in a table of size n
  h: K —> {0, 1, …, n-2, n-1}
- A good hash function should:
  - Minimize collisions
  - Be easy and quick to compute
  - Distribute key values evenly amongst the buckets
  - Use all the information provided in the key

Locality Sensitive Hashing (LSH)

- General idea: Use a hash function that tells whether x and y is a candidate pair (a pair of elements whose similarity must be evaluated)

- A hash function, h, is LSH if it satisfies for some similarity function d:

  - P(h(x) = h(y)) is high if $D(\mathbf{x}, \mathbf{y}) \leq r, r > 0$

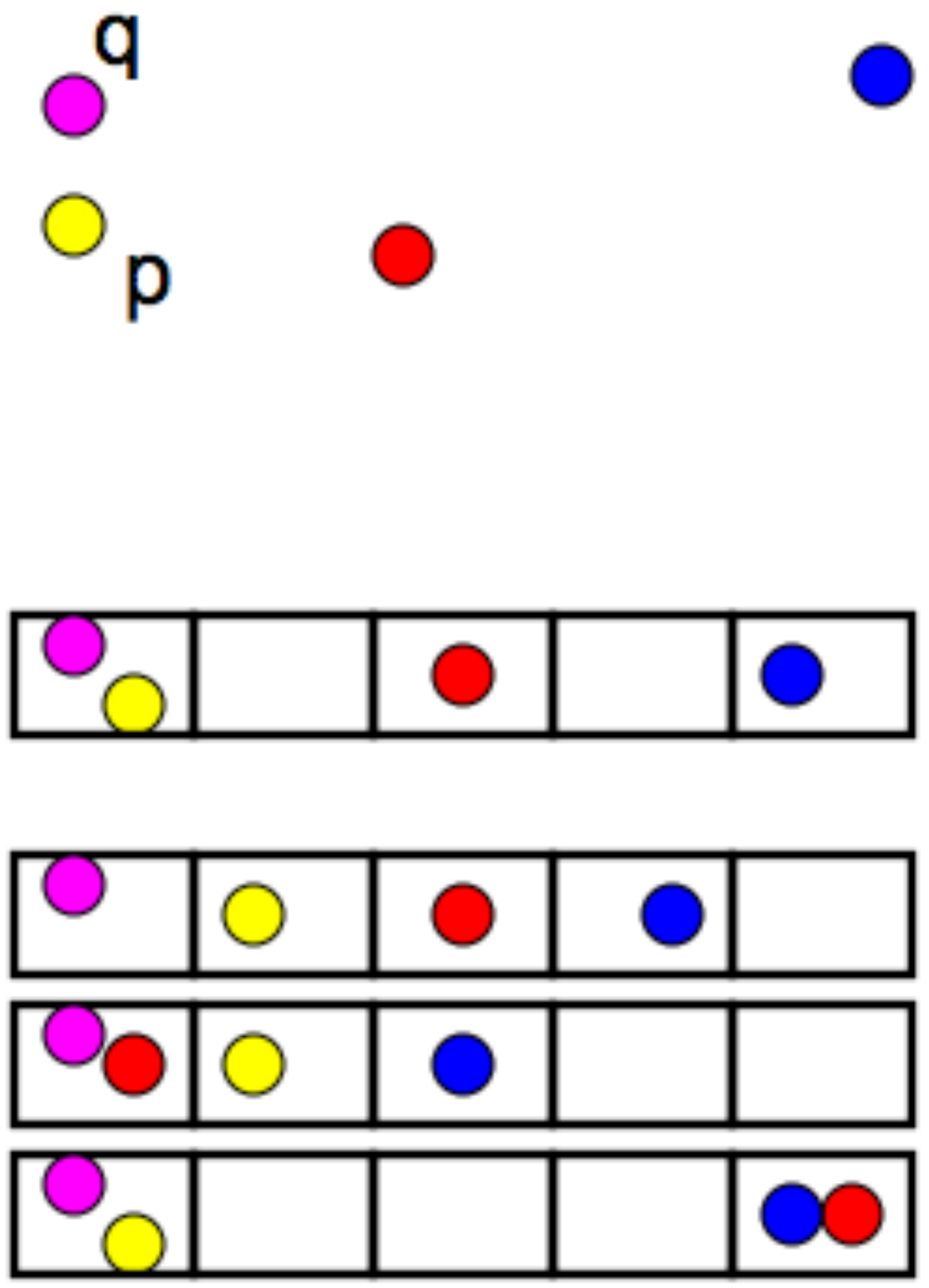
  - P(h(x) = h(y) is low if $D(\mathbf{x}, \mathbf{y}) > \alpha r, r > 0, \alpha > 0$

  - (in between, not sure about probability)

http://courses.cs.washington.edu/courses/cse599c1/13wi/slides/lsh-hashkernels-annotated.pdf

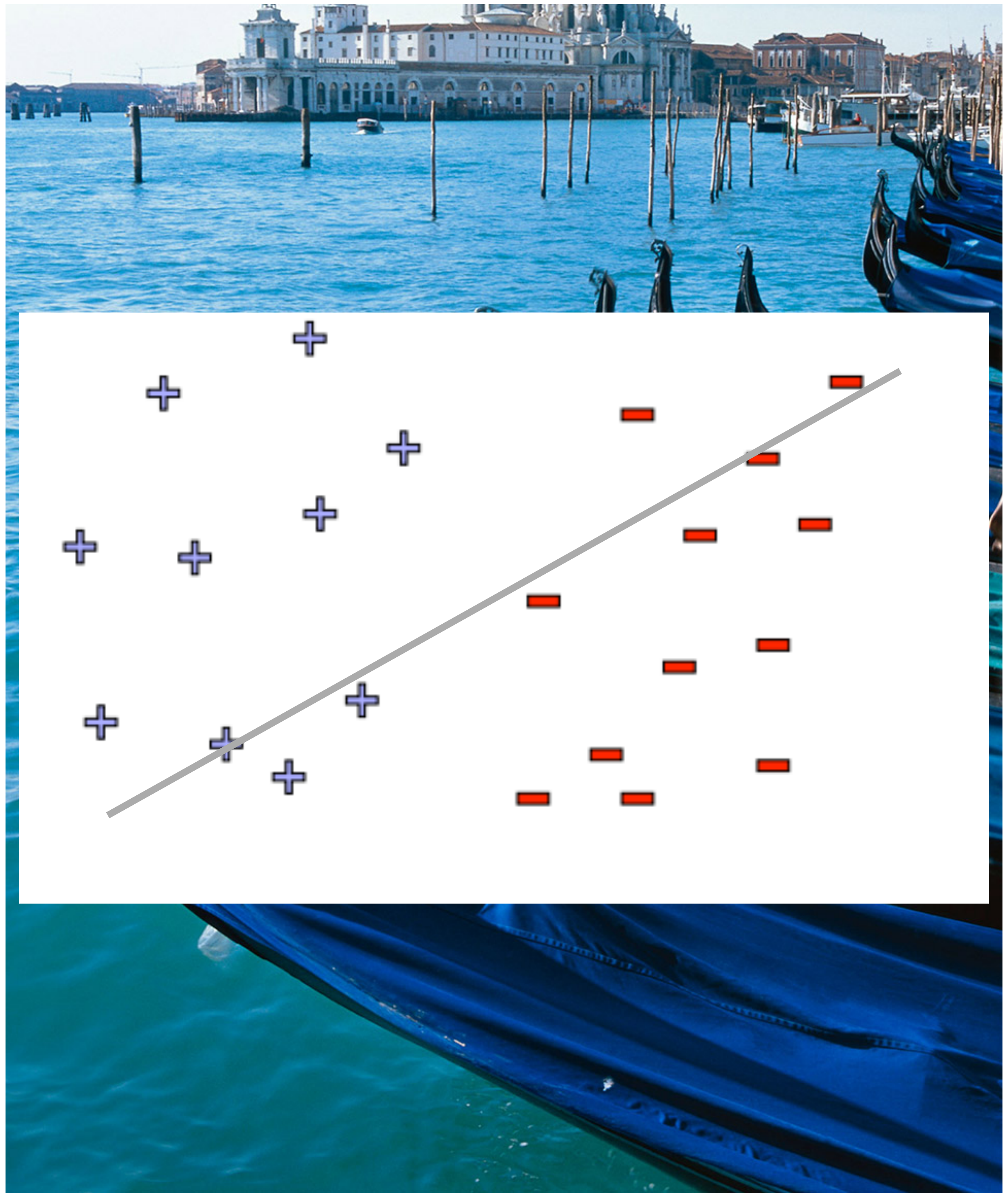
LSH [Indyk-Motwani, 1998]

- Family of hash functions
    - Close points to same buckets
    - Faraway points to different buckets
- Idea: Only examine those items where the buckets are shared
- (Pro) Designed correctly, only a small fraction of pairs are examined
- (Con) There maybe false negatives



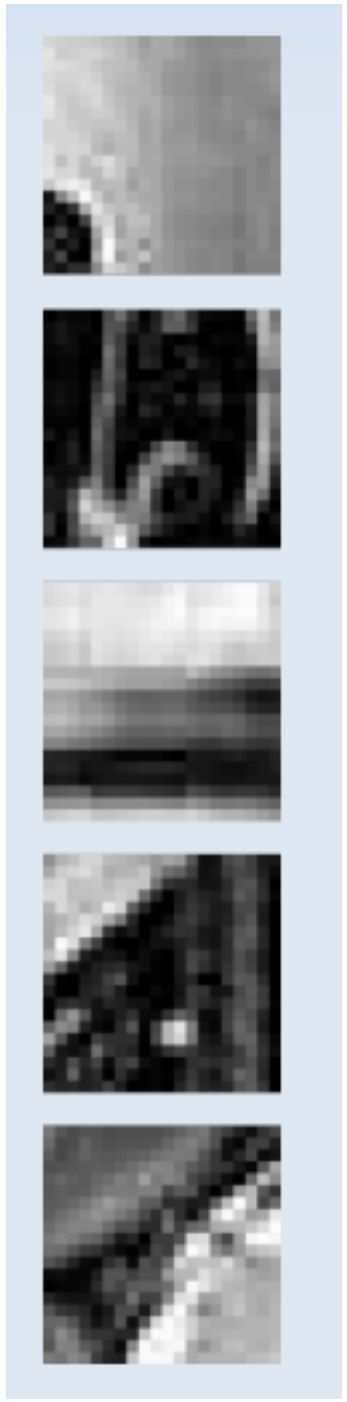
Random Projection Illustrated

- Pick a random vector v using independent Gaussian coordinates
- Project the points onto this random vector
- For most vectors, it will preserve some notion of separability



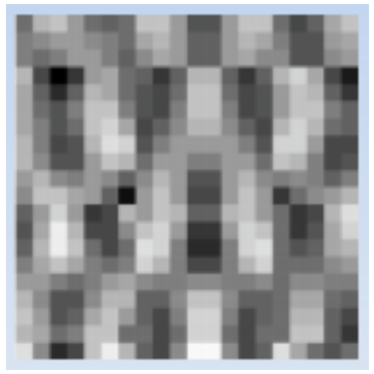
Experiment: Motorcycle Images

- 59,500 20x20 patches taken from motorcycle images
- Each image is represented as 400-dimensional column vectors
- Convert feature vectors into binary strings and use Hamming hash functions

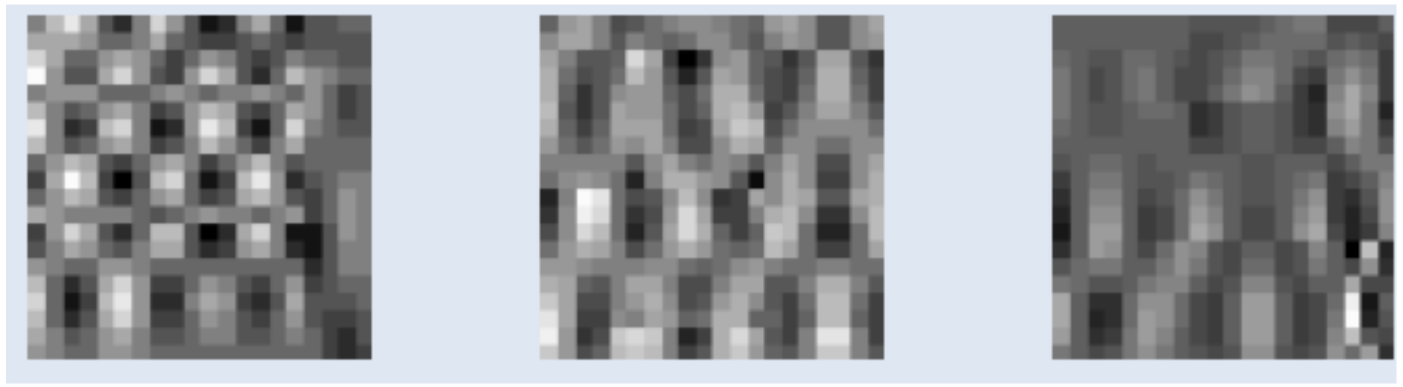


Experiment: Motorcycle Example Query

- Query =



- Examples searched: 7,722 of 59,500
- Result =



- Exact NN =