Tufts

CS135
Introduction to Machine Learning

Lecture 16: Efficiently Finding Neighbors & Uses of Nearest-Neighbors Models

Review: Finding Nearest Neighbors

- Naïve implementations of these measures can be problematic
- For n dimensions each comparison of two points requires $\mathrm{O}(n)$ operations, which is *often* reasonable
- *However*, the classifications work best when we have large amounts of data relative to the number of dimensions
  - Ideally, we have $\mathrm{O}(2^n)$ input points
  - Much smaller numbers tend to lead to poor classifications due to large numbers of outliers.
- *However*, if we compare all pairs of points, we have $\mathrm{O}(|X|^2)$ such operations, where $|X|$ is total size of data-set
  - This can be much too cumbersome for large data-sets

KD-Trees: Efficient Neighbor Calculation

function BUILD-TREE (X, d) **returns a tree**

inputs: $X = \{\mathbf{x}_1 \ldots, \mathbf{x}_m\}$, a set of n-dimensional data-points, and depth d

local variables: $S \geq 1$, a pre-set size limit for sets

if $|X| \leq S$:

return : $Node(X)$, a tree-node containing all elements of X

else :

$\delta \leftarrow (d \bmod n) + 1$ (the dimension for splitting inputs)

$m_\delta \leftarrow$ the median for dimension δ in X

$X^- \subseteq X \leftarrow$ the set of all data-points $\mathbf{x}_i \leq m_\delta$ for dimension δ

$X^+ \subseteq X \leftarrow$ the set of all data-points $\mathbf{x}_j > m_\delta$ for dimension δ

$\mathbf{N}^\circ \leftarrow Node(m_\delta)$, a tree-node containing median-value m_δ

$\mathbf{N}^\circ_{left} \leftarrow$ BUILD-TREE $(X^-, d+1)$

$\mathbf{N}^\circ_{right} \leftarrow$ BUILD-TREE $(X^+, d+1)$

return : $\mathbf{N}^\circ$

- We can build a data-structure to search for nearest neighbors efficiently
- A recursive algorithm, called on original data set, X:

BUILD-TREE$(X, 0)$

K-D Trees: Efficient Neighbor Calculation

function BUILD-TREE(X, d) **returns a tree**

inputs: $X = \{\mathbf{x}_1 \ldots, \mathbf{x}_m\}$, a set of n-dimensional data-points, and depth d

local variables: $S \geq 1$, a pre-set size limit for sets

if $|X| \leq S$:

return : $Node(X)$, a tree-node containing all elements of X

else :

$\delta \leftarrow (d \bmod n) + 1$ (the dimension for splitting inputs)

$m_\delta \leftarrow$ the median for dimension δ in X

$X^- \subseteq X \leftarrow$ the set of all data-points $\mathbf{x}_i \leq m_\delta$ for dimension δ

$X^+ \subseteq X \leftarrow$ the set of all data-points $\mathbf{x}_j > m_\delta$ for dimension δ

$\mathbf{N}^\circ \leftarrow Node(m_\delta)$, a tree-node containing median-value m_δ

$\mathbf{N}^\circ_{left} \leftarrow$ BUILD-TREE$(X^-, d+1)$

$\mathbf{N}^\circ_{right} \leftarrow$ BUILD-TREE$(X^+, d+1)$

return : $\mathbf{N}^\circ$

Each time we go deeper down the tree, we cycle to the next data feature

- Input parameter d sets the feature we use to divide data into separate subsets
- We cycle through these features: $x_1 \rightarrow x_2 \rightarrow \cdots \rightarrow x_{n-1} \rightarrow x_n \rightarrow x_1 \rightarrow \cdots$

K-D Trees: Efficient Neighbor Calculation

function BUILD-TREE(X, d) **returns a tree**

inputs: $X = \{\mathbf{x}_1 \ldots, \mathbf{x}_m\}$, a set of n-dimensional data-points, and depth d

local variables: $S \geq 1$, a pre-set size limit for sets

if $|X| \leq S$:

return : $Node(X)$, a tree-node containing all elements of X

else :

$\delta \leftarrow (d \bmod n) + 1$ (the dimension for splitting inputs)

$m_\delta \leftarrow$ the median for dimension δ in X

$X^- \subseteq X \leftarrow$ the set of all data-points $\mathbf{x}_i \leq m_\delta$ for dimension δ

$X^+ \subseteq X \leftarrow$ the set of all data-points $\mathbf{x}_j > m_\delta$ for dimension δ

$\mathbf{N}^\circ \leftarrow Node(m_\delta)$, a tree-node containing median-value m_δ

$\mathbf{N}^\circ_{left} \leftarrow$ BUILD-TREE$(X^-, d+1)$

$\mathbf{N}^\circ_{right} \leftarrow$ BUILD-TREE$(X^+, d+1)$

return : $\mathbf{N}^\circ$

Data divides along the median value of the chosen feature

- Once a feature is chosen, we find the median value for that feature, and divide all data in two at that median point

K-D Trees: Efficient Neighbor Calculation

function $\textsc{Build-Tree}\,(X,\,d)$ **returns a tree**

inputs: $X = \{\mathbf{x}_1 \ldots, \mathbf{x}_m\}$, a set of n-dimensional data-points, and depth d

local variables: $S \geq 1$, a pre-set size limit for sets

if $|X| \leq S$:

return : $Node(X)$, a tree-node containing all elements of X

else :

$\delta \leftarrow (d \bmod n) + 1$ (the dimension for splitting inputs)

$m_\delta \leftarrow$ the median for dimension δ in X

$X^- \subseteq X \leftarrow$ the set of all data-points $\mathbf{x}_i \leq m_\delta$ for dimension δ

$X^+ \subseteq X \leftarrow$ the set of all data-points $\mathbf{x}_j > m_\delta$ for dimension δ

$\mathbf{N}^\circ \leftarrow Node(m_\delta)$, a tree-node containing median-value m_δ

$\mathbf{N}^\circ_{left} \leftarrow \textsc{Build-Tree}\,(X^-,\, d+1)$

$\mathbf{N}^\circ_{right} \leftarrow \textsc{Build-Tree}\,(X^+,\, d+1)$

return : $\mathbf{N}^\circ$

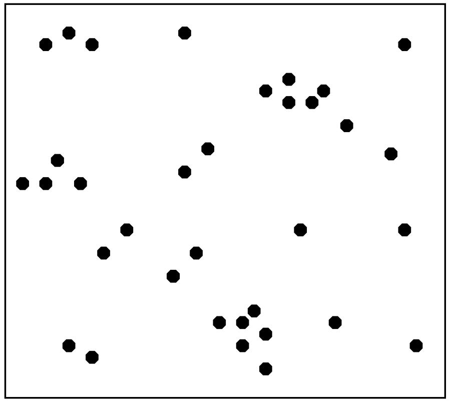
Recursion ends when data subsets are small enough

Recursive calls build a binary tree, branch by branch

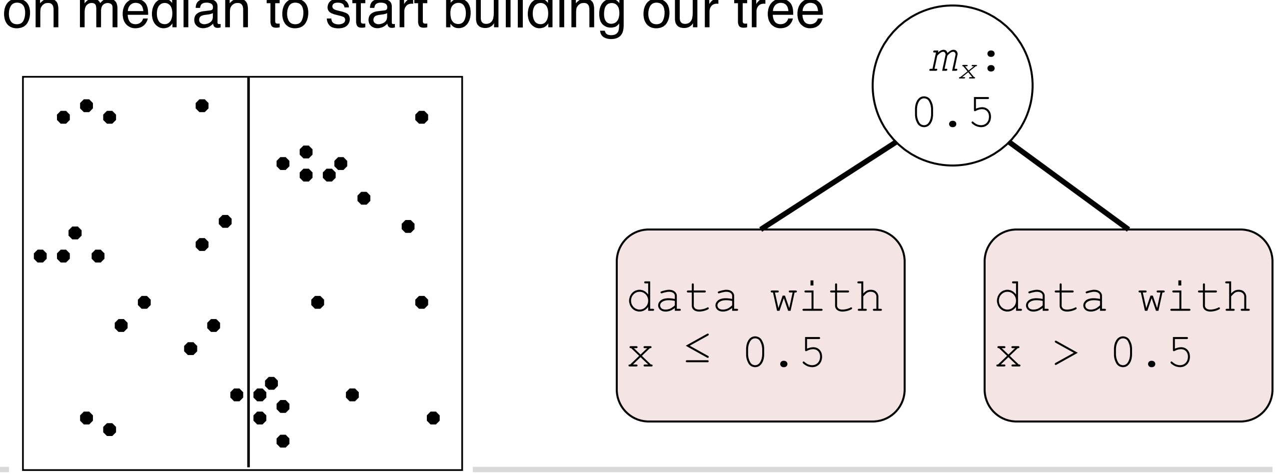
- Recursively builds a binary tree, with sub-tree roots each containing a median value
- Recursion terminates whenever we hit a pre-determined minimum data-set size

A 2-Dimensional Example

- We start with a set of 2-dimensional data-points, $p_i = (x_i, y_i)$

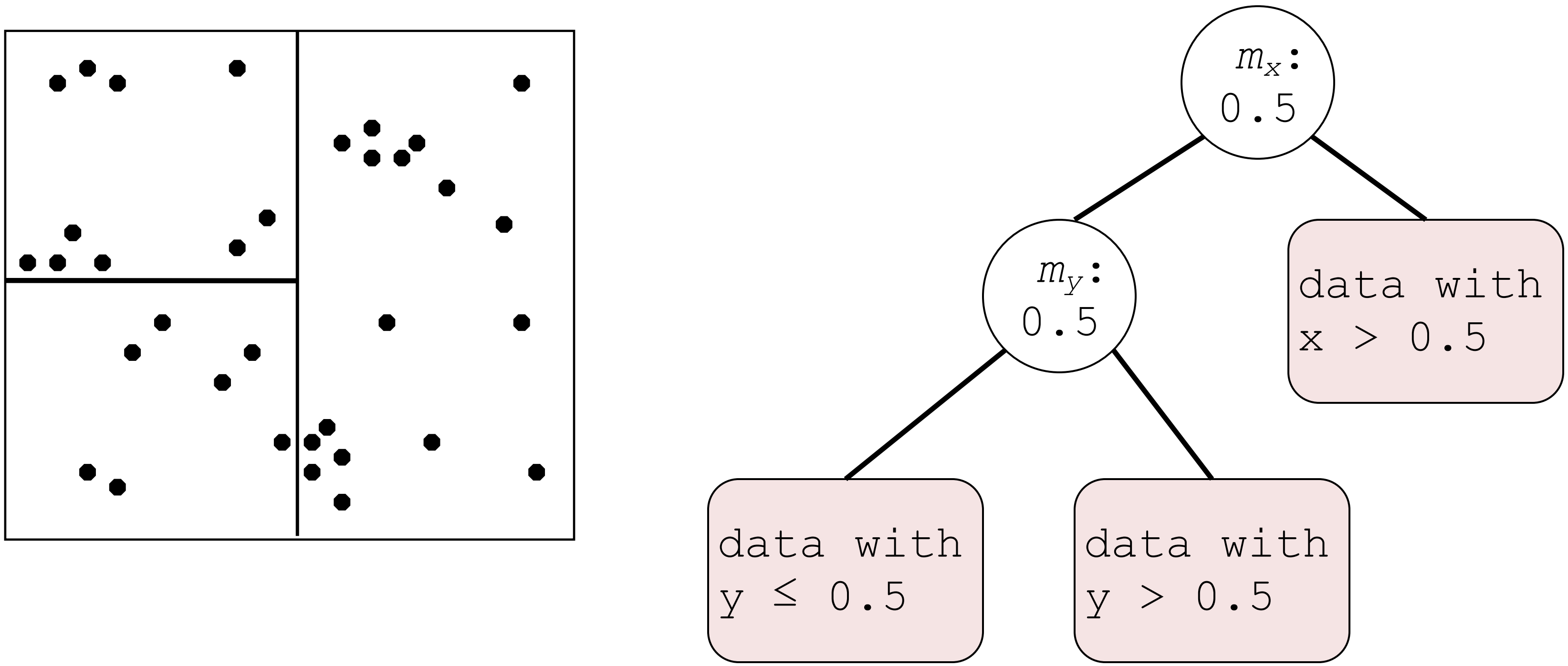


- Split along x-dimension median to start building our tree



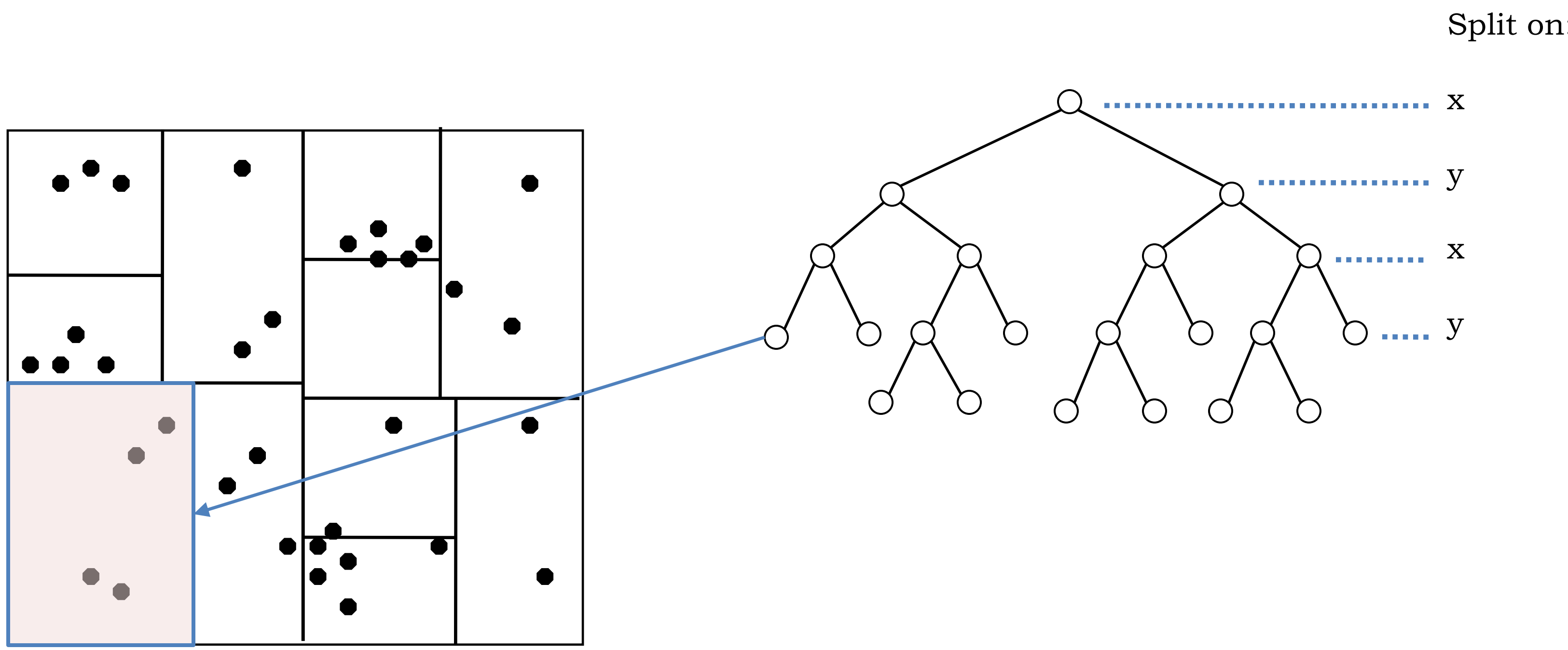
A 2-Dimensional Example

- We then split each group along the y-dimension median separately.



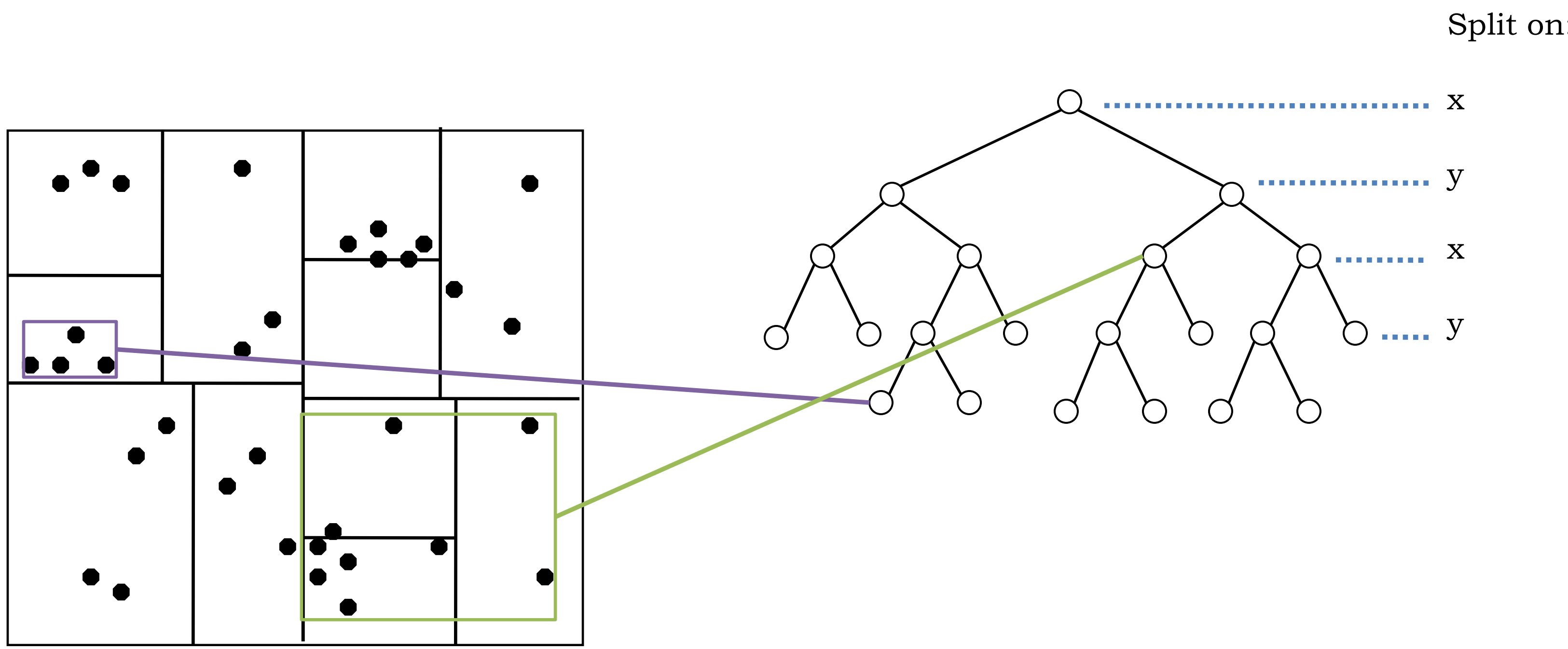
- We repeat on the other branch and continue in each case, *splitting again* on dimensions $x, y, x, y, x, y, \ldots$

A 2-Dimensional Example



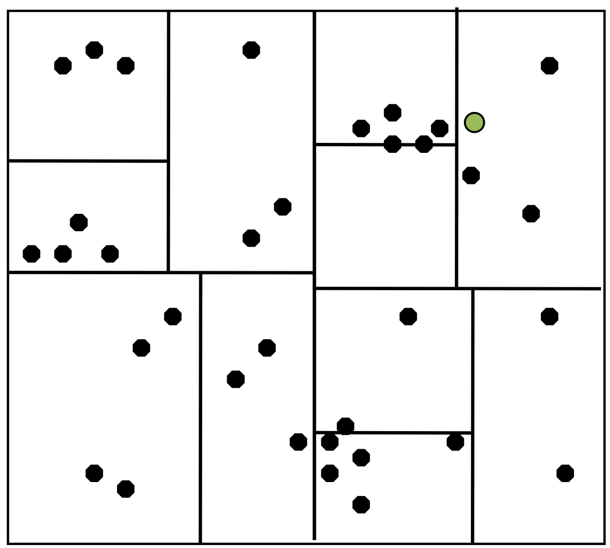
- We stop when we have small enough subsets, each of which is stored in and represented by a leaf-node of our tree
- Interior nodes store median values

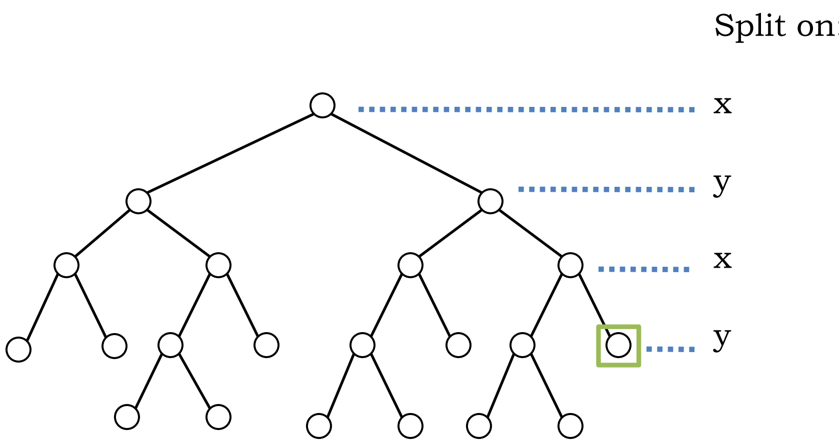
A 2-Dimensional Example



- Interior nodes store median values
- Each node (leaf ***or*** interior) also stores information about the least (tightest) bounding box of all points below it in its sub-tree

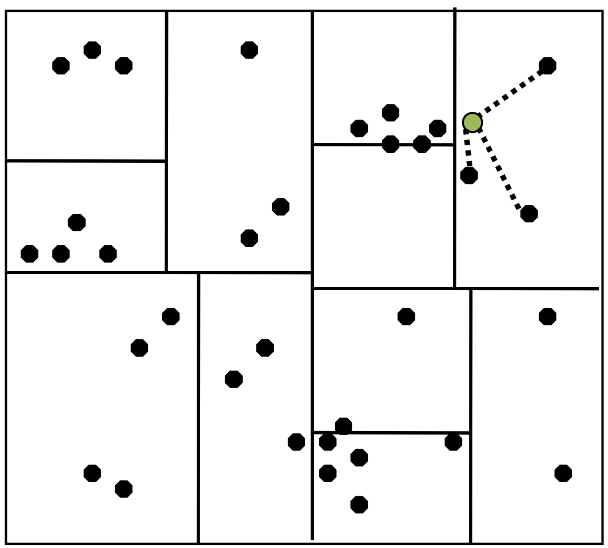
Querying the Tree for the Nearest Neighbor

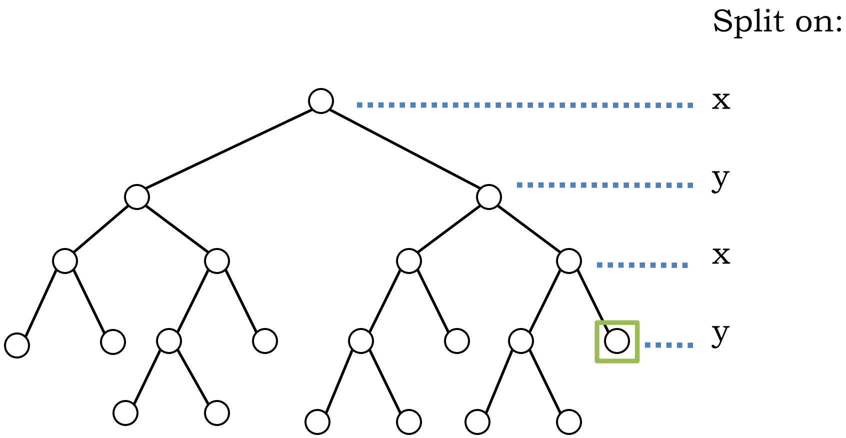




- Suppose we want to find the nearest neighbor of a new data-point (red)
- We start by isolating what sub-set it belongs to, following branches according to the median values (like a binary search tree)

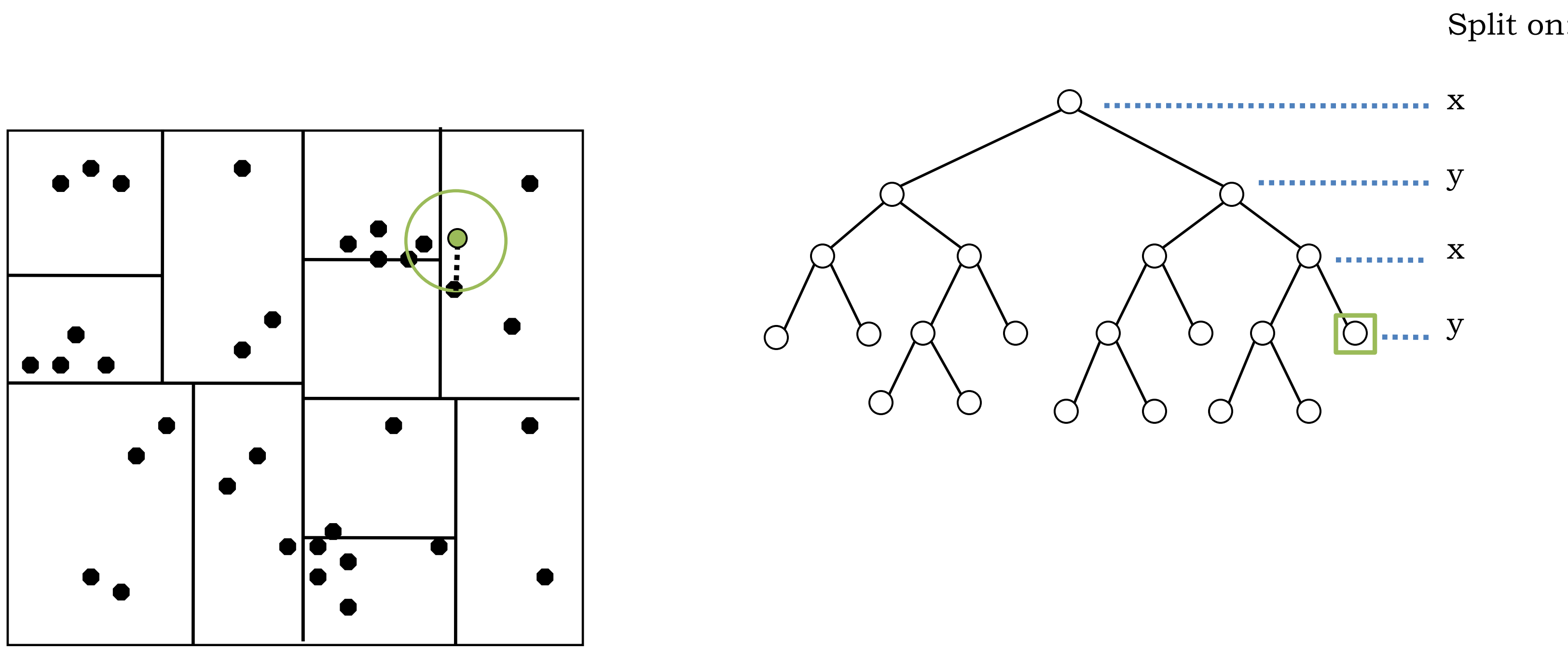
Querying the Tree for the Nearest Neighbor





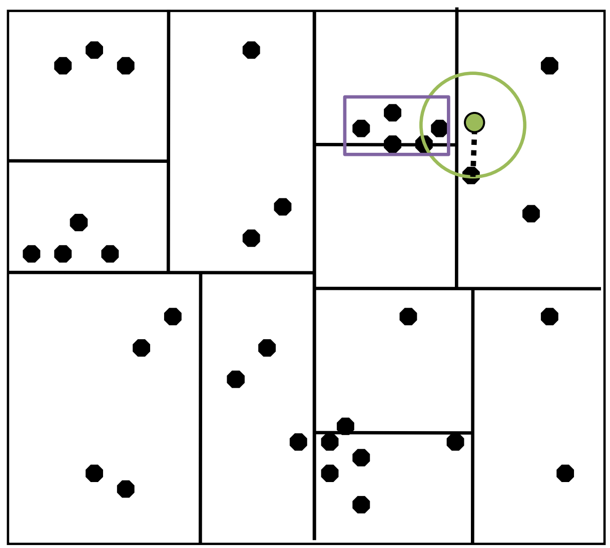
- Once we have found the proper subset, we measure all distances within it
- The closest neighbor *may be* in this set, but it *may not*

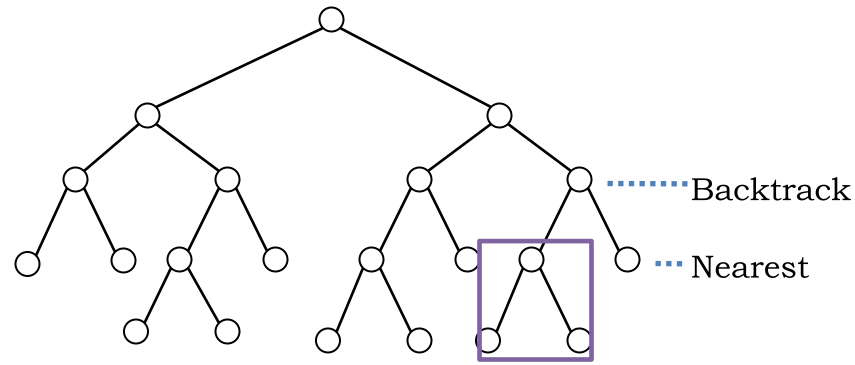
Querying the Tree for the Nearest Neighbor



- We need to check any data-point that *could be* closer to our new point
- The tree helps us here, as we can do some pruning as we go backwards up the tree towards the root

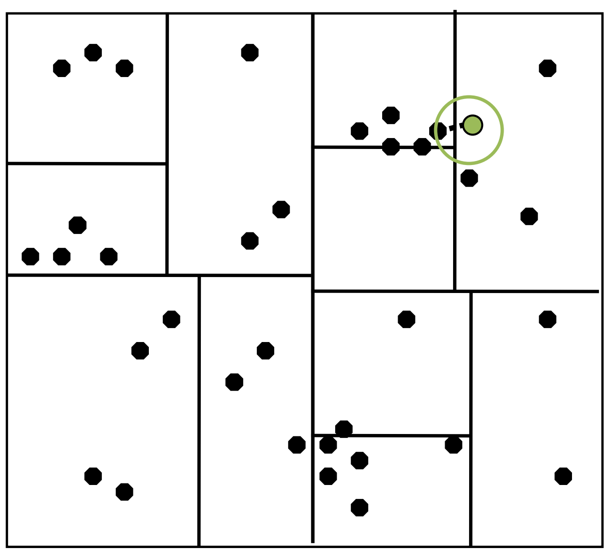
Querying the Tree for the Nearest Neighbor

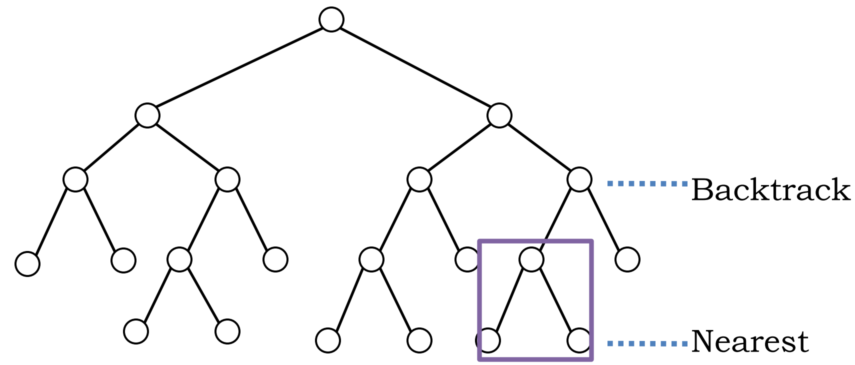




- As we back-track up the tree, we check any branch where the stored bounding box intersects our current bounds

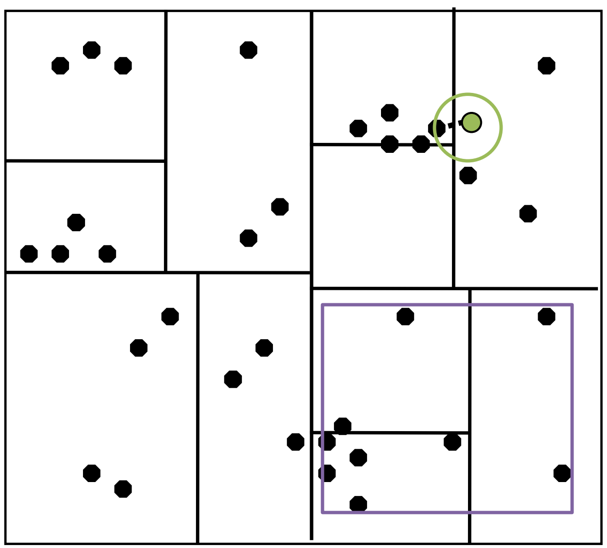
Querying the Tree for the Nearest Neighbor

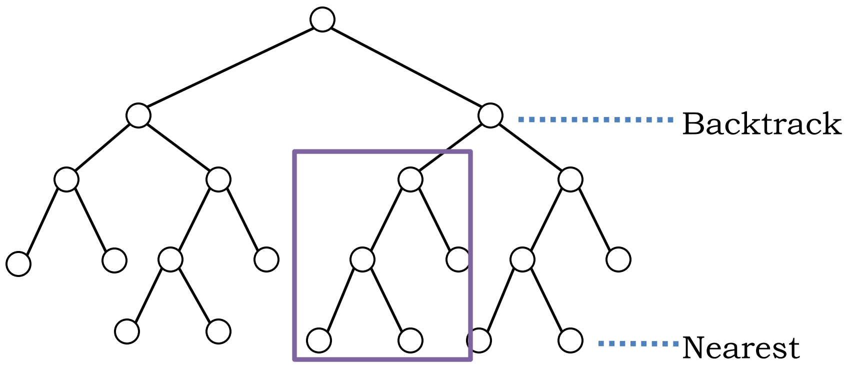




- When this happens, we compute distances to all required nodes, and update distance measure if necessary

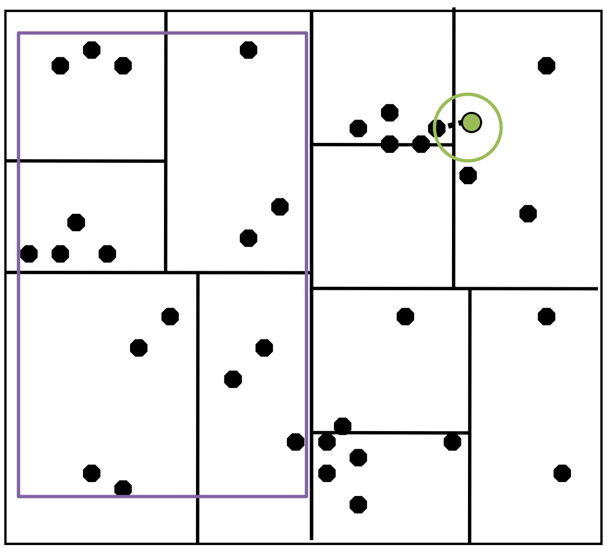
Querying the Tree for the Nearest Neighbor

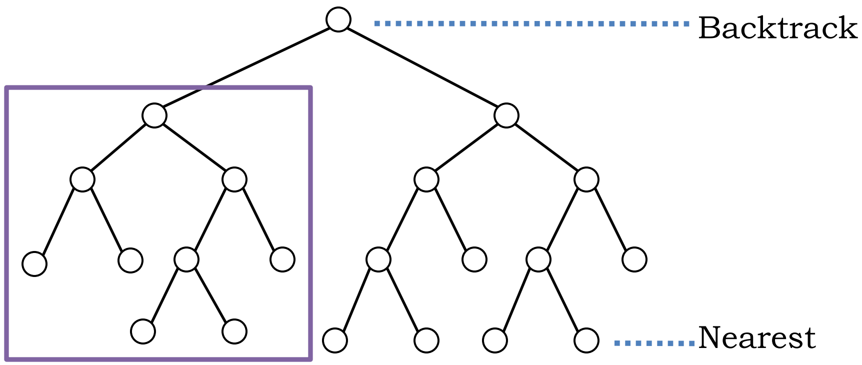




- When we see a sub-tree with a bounding box that *does not* intersect our current bound, we can *ignore it*, saving time overall

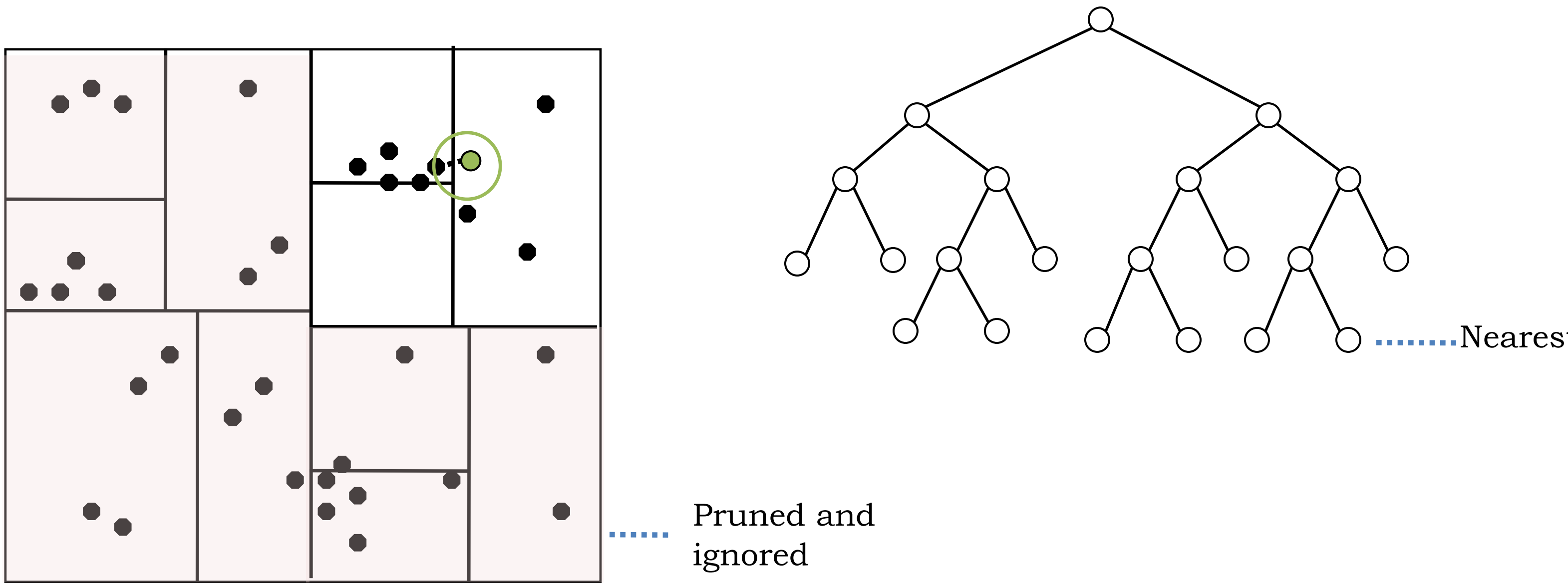
Querying the Tree for the Nearest Neighbor





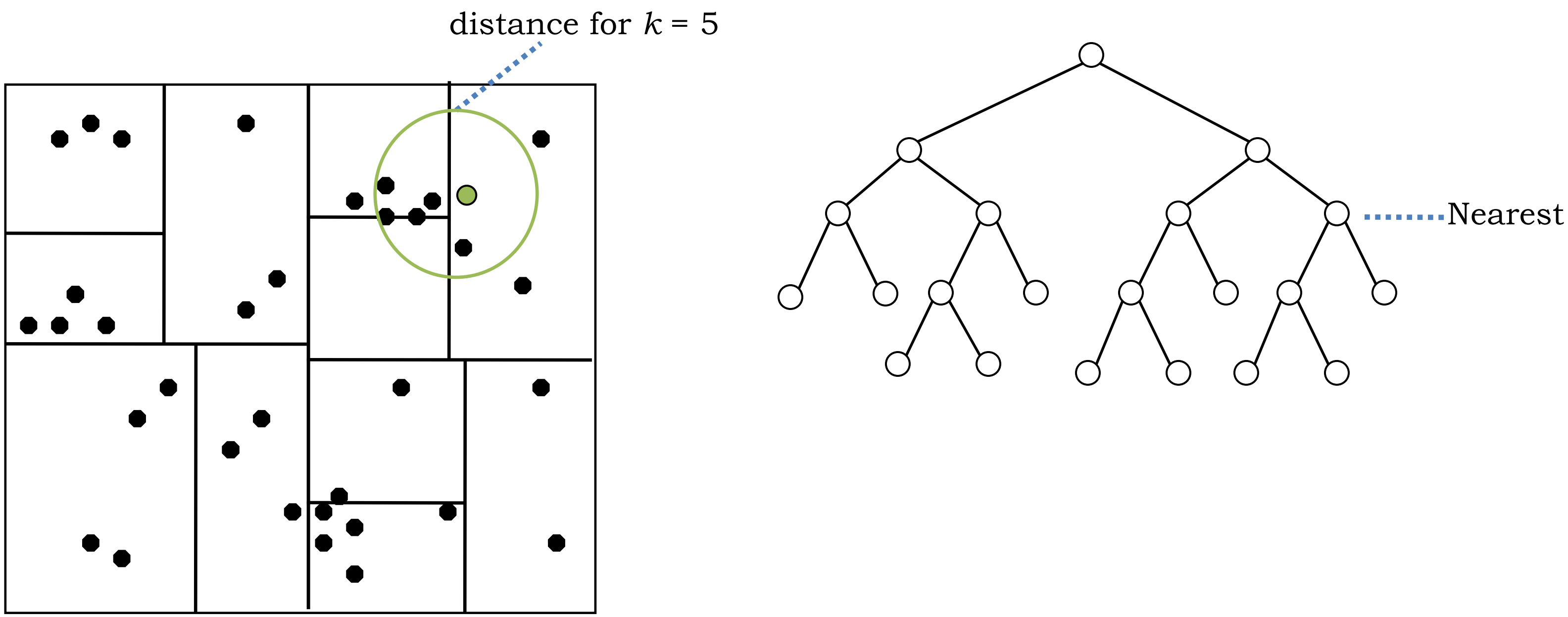
- When we see a sub-tree with a bounding box that *does not* intersect our current bound, we can *ignore it*, saving time overall
- Once we are at the root, we have found the overall nearest neighbor

Querying the Tree for the Nearest Neighbor



- The data structure may allow us to prune off large nodes, restricting those we need to measure the distance from a new point.
- Although it is possible that we still have to do $O(N)$ comparisons, under many distributions of data points, we get $O(\log N)$, significantly speeding up our algorithm for classification.

K-D Trees for *k*-Nearest Neighbors



- If we want not the single nearest neighbor point but some set of k such points (for better classification), the *same approach* can be used.
- It works the same way, but the distance measure is set to use the full set of neighbors (i.e., distance to the *farthest one* of the k nearest)

Review: Uses of Nearest Neighbors

- Once we have found the k-nearest neighbors of a point, we can use this information:

1. *In and of itself*: sometimes we just want to know what those nearest neighbors are (items that are similar to a given piece of data)
2. *For additional classification purposes*: we want to find the nearest neighbors in a set of *already-classified* data and then use those neighbors to classify new data.
3. *For regression purposes*: we want to find the nearest neighbors in a set of points for which we *already know* a functional (scalar) output and then use those outputs to generate the output for some new data.

Measuring Distances for Document Clustering & Retrieval

- Suppose we want to rank documents in a database or on the web-based on how similar they are
  - We want a distance measurement that relates to them
  - We can do a nearest-neighbor query for any article to get a set of those that are the closest (and most similar)
  - Searching for additional information based on a given document is equivalent to finding its nearest neighbors in the set of all document.

The "Bag of Words" Document Model

- Suppose we have a set of documents $X = \{x_1, x_2, \dots, x_n\}$
- Let $W = \{w \mid w \text{ is a word in some document } x_i\}$
- We can then treat each document x_i as a vector of word counts (how many times each word occurs in the document):

$$C_i = \{c_{i,1}, c_{i,2}, \dots, c_{i,|W|}\}$$

  - Assuming some fixed order of the set of words W
  - Not every word occurs in every document, so some count values may be set to 0

- As previously noted, values tend to work better for purposes of classification if they are *normalized*, so we set each value to be between 0 and 1 by dividing on the largest count seen for *any* word in *any* document:

$$c_{i,j} \leftarrow \frac{c_{i,j}}{\max_{k,m} c_{k,m}}$$

Tufts

Distances between Words

- We can now compute the distance function between any two documents (here, we use the Euclidean):

$$d(x_i, x_j) \;=\; \sqrt{\sum_{k=1}^{|W|} (c_{i,k} - c_{j,k})^2}$$

- We could then build a KD-Tree, using the vectors of words as our dimension values, and query for some set of most similar documents to any document we start with
- *Problem*: word counts turn out to be a lousy metric!
  - Common everyday words dominate the counts, making most documents appear quite similar and making retrieval poor.

Better Measures of Document Similarity

- We want to emphasize *rare words* over common ones:

1. Define word frequency: $t(w,x)$ as the (normalized) count of occurrences of word w in document x

$$c_x(w) = \#\text{ times word } w \text{ occurs in document } x$$

$$c_x^\star = \max_{w \in W} c_x(w)$$

$$t(w,x) = \frac{c_x(w)}{c_x^\star}$$

2. Define inverse document frequency of word w:

$$id(w) = \log \frac{|X|}{1 + |\{x \in X \mid w \in x\}|}$$

Total # of documents

that contain word w

3. Use combined measure for each word and document:

$$tid(w,x) = t(w,x) \times id(w)$$

Inverse Document Frequency

- We want to emphasize *rare words* over common ones:

$$
\begin{aligned}
id(w) &= \log \frac{|X|}{1 + |\{x \in X \mid w \in x\}|} \\
tid(w,x) &= t(w,x) \times id(w)
\end{aligned}
$$

- $id(w)$ goes to 0 as the word w becomes more common
- $tid(w,x)$ is highest when w occurs *often* in document x, but is *rare overall* in the full document set

An Example

- The inverse document frequency of word w:

$$id(w) \;=\; \log \frac{|X|}{1 + |\{x \in X \mid w \in x\}|}$$

- Suppose we have $1{,}000$ documents ($|X| = 1000$), and the word *the* occurs in every single one of them:

$$id(the) = \log \frac{1000}{1001} \approx -0.001442$$

- Conversely, if the word *banana* only appears in 10 of them:

$$id(banana) = \log \frac{1000}{10} \approx 6.644$$

- Thus, when calculating normalized word counts, the *banana* gets treated as being about 4,600 times more important than *the*!
  - If we threshold $id(w)$ to a minimum of 0 (never negative), we then ***completely ignore*** words that are in every document

Distances between Words

- Given the threshold on the inverse document frequency, the distance between two documents is now ***proportional*** to that measure:

$$d(x_i, x_j) = \sqrt{\sum_{k=1}^{|W|} (tid(w_k, x_i) - tid(w_k, x_j))^2}$$

$$= \sqrt{\sum_{k=1}^{|W|} ([t(w_k, x_i) \times id(w_k)] - [t(w_k, x_j) \times id(w_k)])^2}$$

$$= \sqrt{\sum_{k=1}^{|W|} (id(w_k) \times [t(w_k, x_i) - t(w_k, x_j)])^2}$$

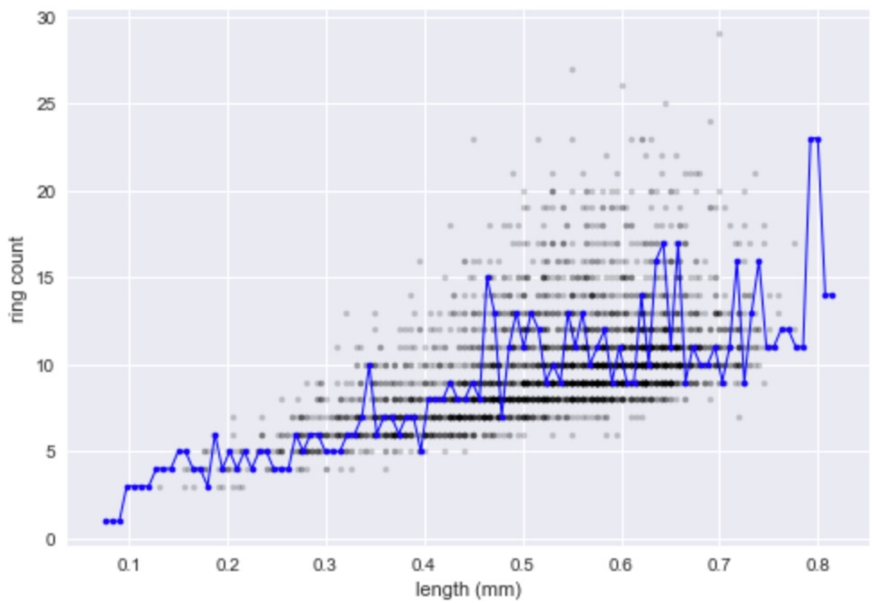
- Our KD-Tree can now efficiently find similar documents based upon this metric
- Mathematically, words for which frequency $id(w) = 0$ have no effect on the distance
  - Obviously, in implementing this we can simply ***remove*** those words from word-set W in the first place to skip useless clock-cycles…

Nearest-Neighbor Regression

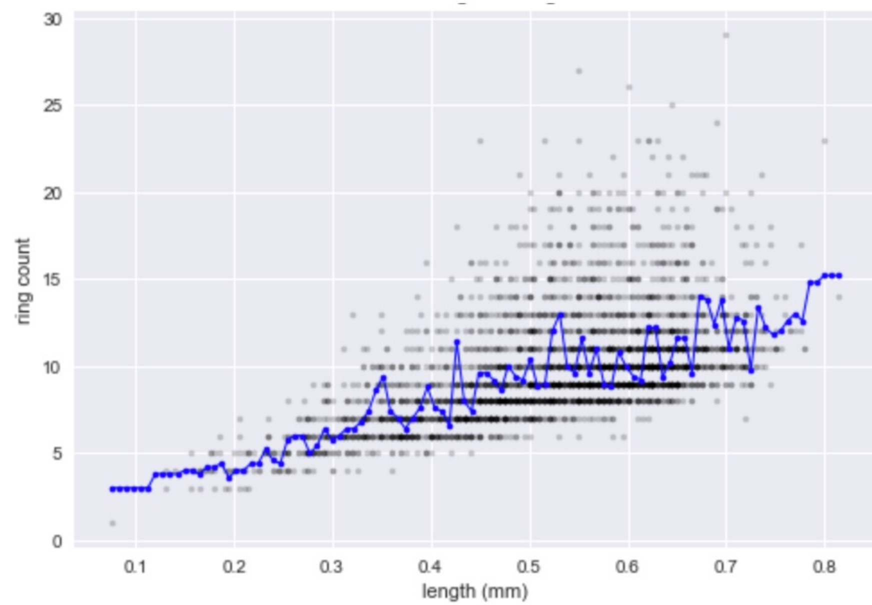
- Given a data-set of various features of abalone (sex, size, weight, etc.), a regression classifier predicts shellfish age
- A training set of measurements, with real age determined by counting rings in the abalone shell, is analyzed and grouped into nearest neighbor units
- A predictor for new data is generated according to the average age value of neighbors

Nearest-Neighbor Regression

1-nearest neighbor



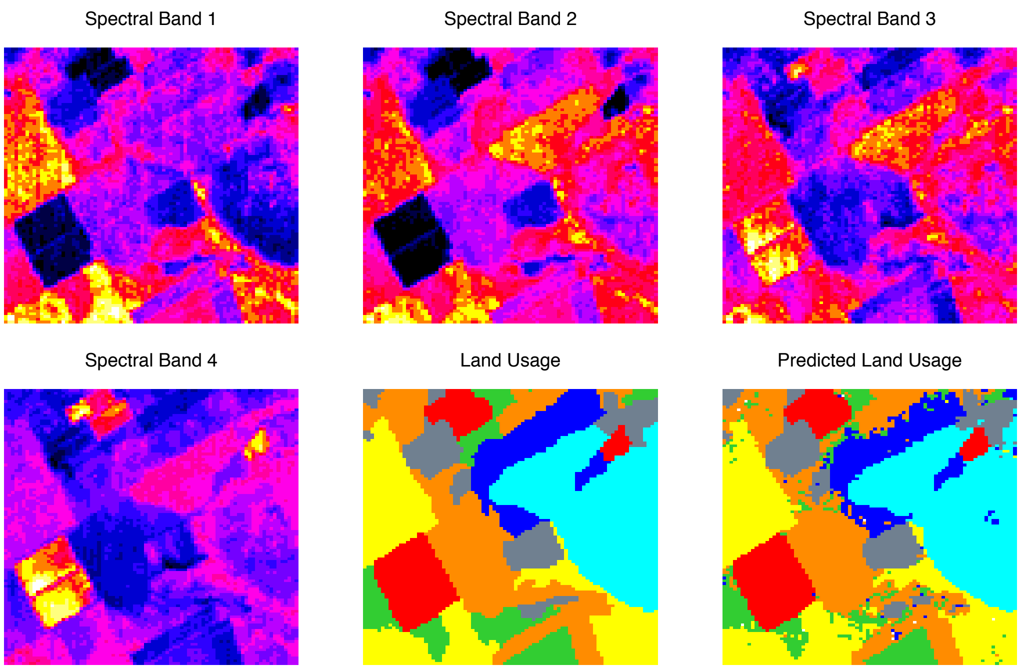
5-nearest neighbors



- Predictions for 100 points, given regression on shell length and age
- With one-nearest neighbor (left), the result has higher variability and predictions are noisier
- With five-nearest neighbors (right), results are smoothed out over multiple data-points

Nearest-Neighbor Clustering for Image Classification

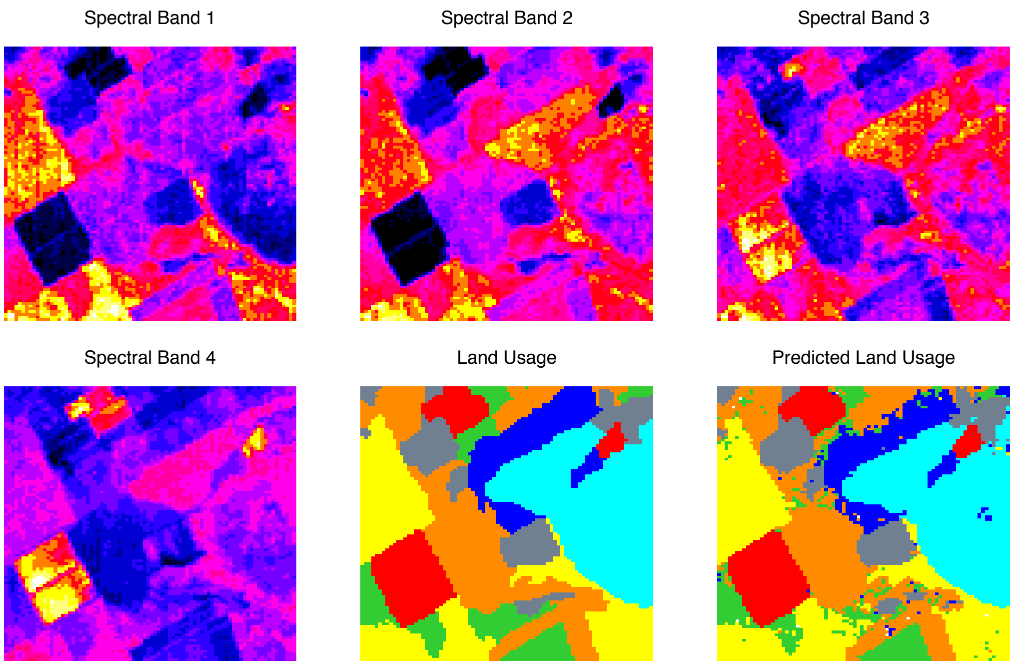
Image source: Hastie, et al., *Elements of Statistical Learning* (Springer, 2017)



- As part of the STATLOG project (Michie et al., 1994): given satellite imagery of land, predict its agricultural use for mapping purposes
- Training set: sets of images in 4 spectral bands, with actual use of land (7 soil/crop categories) based upon manual survey

Nearest-Neighbor Clustering for Image Classification

Image source: Hastie, et al., *Elements of Statistical Learning* (Springer, 2017)



N	N	N
N	X	N
N	N	N

- To predict the usage for a given pixel in a new image:
  1. In each band, get value of a pixel and 8 adjacent, for (4 x 9) = 36 features
  2. Find the 5 nearest neighbors of that feature-vector in labeled training set
  3. Assign the land use class of the majority of those 5 neighbors
- Achieved test error of 9.5% with a very simple algorithm