Case Study 1: Estimating Click Probabilities

Tackling an Unknown Number of Features with Sketching

Machine Learning/Statistics for Big Data
CSE599C1/STAT592, University of Washington

Emily Fox ~~Carlos Guestrin~~

January 22nd, 2013

Sketching Counts

- Bloom Filter is super cool, but not what we need…
  - We don't just care about whether a feature existed before, but to keep track of counts of occurrences of features!
- Recall Perceptron update:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \mathbb{1}\left[y^{(t)}(\mathbf{w}^{(t)} \cdot \mathbf{x}^{(t)}) \leq 0\right] y^{(t)}\mathbf{x}^{(t)}$$

- Must keep track of counts of each feature (weighed by $y^{(t)}$):
  - E.g., with sparse data, for each non-zero dimension *i* in $\mathbf{x}^{(t)}$:

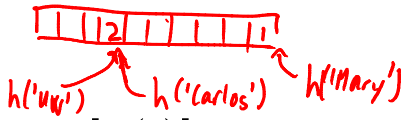
If mistake + $\forall\, x_i^{(t)} \neq 0$

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + y^{(t)} x_i^{(t)}$$

- Can we generalize the Bloom Filter?

Count-Min Sketch: single vector

- Simpler problem: Count how many times you see each string
- Single hash function: h
  - Keep Count vector of length m
  - every time see string i:



$$Count[h(i)] \leftarrow Count[h(i)] + 1$$

See 'Carlos' 'uw' $\Rightarrow$ Count[4] = 2

Q('Carlos') = Count[4] = 2 > 1 true counts

  - Again, collisions could be a problem:
    - a_i is the count of element i:

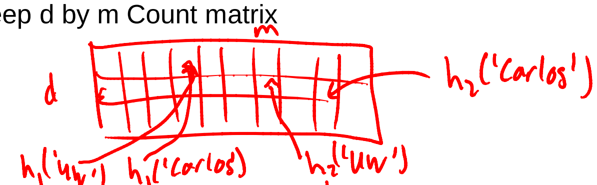
$$Count[j] = \sum_{i: h(i)=j} a_i$$

over-est

$Q(i) \rightarrow$ return $\hat{a}_i = Count[h(i)] \geq a_i$

Count-Min Sketch: general case

- Keep d by m Count matrix



- d hash functions: $h_1, \dots, h_d$
  - Just like in Bloom Filter, decrease errors with multiple hashes
  - Every time see string i:

$$\forall j \in \{1, \dots, d\} : Count[j, h_j(i)] \leftarrow Count[j, h_j(i)] + 1$$

Querying the Count-Min Sketch

$$\forall j \in \{1, \ldots, d\} : Count[j, h_j(i)] \leftarrow Count[j, h_j(i)] + 1$$

- Query Q(i)?
  - What is in Count[j,k]?

    $Count[j,k] = \sum_{i : h_j(i) = k} a_i$
  - Thus: $Q(i)$

    each $Count[j, h_j(i)] \geq a_i$
  - Return:

    $\hat{a}_i \triangleq \min_j Count[j, h_j(i)] \geq a_i$

    tightest upper bound

Analysis of Count-Min Sketch

$$\hat{a}_i = \min_j Count[j, h(i)] \geq a_i$$

- Set:

$$m = \left\lceil \frac{e}{\epsilon} \right\rceil \qquad d = \left\lceil \ln \frac{1}{\delta} \right\rceil$$

- Then, after seeing n elements:

$$a_i \leq \hat{a}_i \leq a_i + \epsilon n$$

not bigger by more than ϵn

- With probability at least 1-δ

Proof of Count-Min for Point Query with Positive Counts: Part 1 – Expected Bound

- $I_{i,j,k}$ = indicator that i & k collide on hash j:

$$(i \neq k) \wedge (h_j(i) = h_j(k))$$

- Bounding expected value:

$$E[I_{ijk}] = P(h_j(i) = h_j(k)) = \frac{1}{m} \leq \frac{\epsilon}{e}$$

- $X_{i,j}$ = total colliding mass on estimate of count of i in hash j:

$$X_{ij} = \sum_{k \neq i} I_{ijk} a_k \qquad Count[j, h_j(i)] = a_i + X_{ij}$$

- Bounding colliding mass:

$$E[X_{ij}] = \overbrace{\sum_{k \neq i} a_k}^{n} \overbrace{E[I_{ijk}]}^{\leq \frac{\epsilon}{e}} \leq \frac{n\epsilon}{e}$$

- Thus, estimate from each hash function is close in expectation

Proof of Count-Min for Point Query with Positive Counts: Part 2 – High Probability Bounds

- What we know: $Count[j, h_j(i)] = a_i + X_{i,j} \qquad E[X_{i,j}] \leq \frac{\epsilon}{e} n$
- Markov inequality: For $z_1,\ldots,z_k$ positive iid random variables

$$P(\forall z_i : z_i > \alpha E[z_i]) < \alpha^{-k}$$

$$P(z_i > a) < \frac{E[z_i]}{a}$$

$$a = \alpha E[z_i]$$

+ ind. of z_i

- Applying to the Count-Min sketch:

$$P(\hat{a}_i > a_i + \epsilon n) = P(\forall_j, count[j, h_j(i)] > a_i + \epsilon n)$$

$$= P(\forall_j, \cancel{a_i} + X_{ij} > \cancel{a_i} + \epsilon n)$$

$$= P(\forall_j \; X_{ij} > e \, E[X_{ij}]) \overset{?}{<} e^{-d} \leq \delta$$

d hash fcns "α" Small prob.

But Our Updates may be positive or Negative

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \mathbb{1}\left[y^{(t)}(\mathbf{w}^{(t)} \cdot \mathbf{x}^{(t)}) \leq 0\right] \underline{y^{(t)}\mathbf{x}^{(t)}}$$

pos. or neg

non-int.

- Count-Min sketch for positive & negative case
  - a_i no longer necessarily positive
- Update the same: Observe change Δ_i to element *i*:

$$\forall j \in \{1, \ldots, d\} : Count[j, h_j(i)] \leftarrow Count[j, h_j(i)] + \Delta_i$$

  - Each Count[j,h(i)] no longer an upper bound on a_i
- How do we make a prediction?

$$\hat{a}_i = \underset{j}{\text{median}}\ \text{Count}[j, h_j(i)]$$

$\text{Count}[1,h_1(i)]$

a_i $\text{Count}[3,h_3(i)]$

$\text{Count}[2,h_2(i)]$

- Bound: $|\hat{a}_i - a_i| \leq \underline{3\epsilon||\mathbf{a}||_1}$
  - With probability at least $1-\delta^{1/4}$, where $||\mathbf{a}|| = \Sigma_i |a_i|$

Finally, Sketching for Perceptron

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \mathbb{1}\left[y^{(t)}(\mathbf{w}^{(t)} \cdot \mathbf{x}^{(t)}) \leq 0\right] \underline{y^{(t)}\mathbf{x}^{(t)}}$$

- Never need to know size of vocabulary! ☆
- Make a mistake, update Count-Min matrix:

$$\forall i \quad x_i^{(t)} \neq 0$$

$$\forall_j \quad \text{Count}[j, h_j(i)] \mathrel{+}= y^{(t)} x_i^{(t)}$$

- Making a prediction:

$$\text{sign}(w^{(t)} \cdot x) = \text{sign}\Big(\sum_i \underbrace{w_i^{(t)}}_{\approx\, w_i^{(t)}} x_i\Big)$$

$$\sum_i w_i^{(t)} x_i \approx \sum_{i: x_i \neq 0} x_i \ \underset{j}{\text{median}}\ \text{count}[j, h_j(i)]$$

- Scales to huge problems, great practical implications… More next time

What you need to know

- Hash functions
- Bloom filter
  - Test membership with some false positives, but very small number of bits per element
- Count-Min sketch
  - Positive counts: upper bound with nice rates of convergence
  - General case
- Application to Perceptron Learning and Prediction

Case Study 2: Document Retrieval

Task Description: Finding Similar Documents

Machine Learning/Statistics for Big Data
CSE599C1/STAT592, University of Washington

Emily Fox

January 22nd, 2013

Document Retrieval

- **Goal:** Retrieve documents of interest
- **Challenges:**
  - Tons of articles out there
  - How should we measure similarity?

Task 1: Find Similar Documents

- **To begin…**
  - **Input:** Query article x
  - **Output:** Set of k similar articles

x

Document Representation

- Bag of words model

ignore order of the words

word count

$$X = \begin{bmatrix} wc_1 \\ wc_2 \\ \vdots \\ wc_d \end{bmatrix}$$

$|V|$ = size of vocab. $= d$

1-Nearest Neighbor

- Articles $\mathcal{X} = \{x^1, \dots, x^N\}$, $x^i \in \mathbb{R}^d$
- Query: x
- 1-NN
  - Goal: find article in $\mathcal{X}$ "closest" to x
  - Formulation: * need distance metric * $d(u,v)$

$$x^{NN} = \underset{x^i \in \mathcal{X}}{\arg\min}\ d(x^i, x)$$

k-Nearest Neighbor

- Articles $X = \{x^1, \ldots, x^N\}, \quad x^i \in \mathbb{R}^d$
- Query: $x \in \mathbb{R}^d$
- k-NN
  - Goal: find k articles in X closest x
  - Formulation:

$$X^{NN} = \{x^{NN_1}, \ldots, x^{NN_k}\} \subseteq X$$

$$\text{s.t. } \forall x^i \in X \setminus X^{NN}$$

$$d(x^i, x) \geq \max_{x^{NN_i} \in X^{NN}} d(x^{NN_i}, x)$$

Distance Metrics – Euclidean

$$d(u,v) = \sqrt{\sum_{i=1}^{d} (u_i - v_i)^2} = \|u-v\|_2$$

Or, more generally, $d(u,v) = \sqrt{\sum_{i=1}^{d} \sigma_i^2 (u_i - v_i)^2}$ weight for dim i

Equivalently,

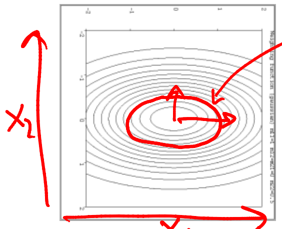
$$d(u,v) = \sqrt{(u-v)'\Sigma(u-v)}$$

where $\Sigma = \begin{bmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & \sigma_d^2 \end{bmatrix}$

Other Metrics…

- Mahalanobis, Rank-based, Correlation-based, cosine similarity…

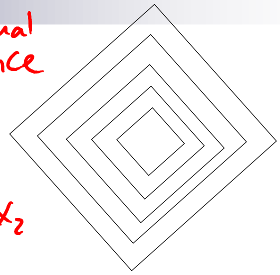
Notable Distance Metrics (and their level sets)



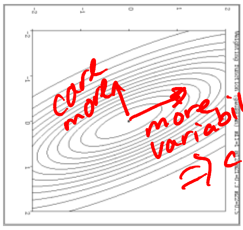
pts of equal distance

care more about changes in x_2

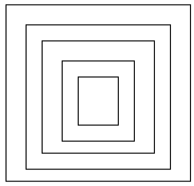
Scaled Euclidian (L_2)



L_1 norm (absolute)



Mahalanobis
(Σ is general sym pos def matrix, on previous slide = diagonal)



$L\infty$ *(max) norm*

Euclidean Distance + Document Retrieval

- Recall distance metric

$$d(u,v) = \sqrt{\sum_{i=1}^{d}(u_i - v_i)^2} = \|u-v\|_2$$

- What if each document were α times longer?
  - Scale word count vectors
    $u \leftarrow \alpha u$
    $v \leftarrow \alpha v$
  - What happens to measure of similarity?

$$\|\alpha u - \alpha v\|_2 = \alpha \|u-v\|_2 > \|u-v\|_2$$

$\alpha > 1$ now less similar

- Good to normalize vectors

$$\|u\|_2 = \|v\|_2 = 1$$

Issues with Document Representation

- Words counts are ***bad*** for standard similarity metrics

rare words swamped by common words

"The koala hugs the tree…" $\longleftrightarrow$ "Tree huggers…"

common words like "tree" and "hug" are dominating (and "the")

- Term Frequency – Inverse Document Frequency (tf-idf)
  - Increase importance of rare words

TF-IDF

- Term frequency:

$$\mathrm{tf}(\underset{\text{term}}{t}, \underset{\text{doc}}{d}) = \#\text{ of occur of } t \in d \triangleq f(t,d)$$

$$\frac{f(t,d)}{\max\{f(w,d) : w \in d\}} \leftarrow \text{prevent bias towards long docs}$$

  - Could also use $\{0,1\}, 1 + \log f(t,d), \ldots$
- Inverse document frequency:

$$\mathrm{idf}(t, \cancel{D}) = \log \frac{|X|}{1 + |\{d \in X : t \in d\}|} \rightarrow 0 \quad t \in \text{many docs } d$$

$$> 0 \quad \text{ow}$$

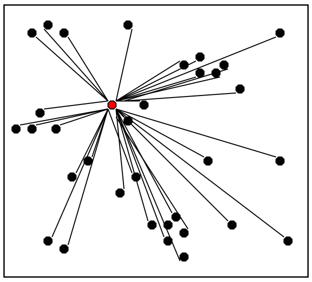
- tf-idf:

$$\mathrm{tfidf}(t, d, \cancel{D}) = tf(t,d) \times idf(t, X)$$

  - High for document d with high frequency of term t (high "term frequency") and few documents containing term t in the corpus (high "inverse doc frequency")

Issues with Search Techniques

- Naïve approach:
  Brute force search
  - Given a query point x
  - Scan through each point x^i
  - O(N) distance computations per 1-NN query!
  - O(Nlog$k$) per k-NN query!

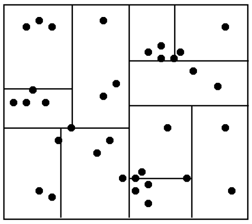


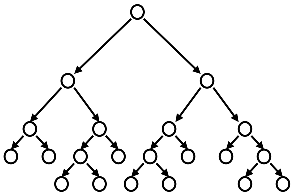
33 Distance Computations

- What if N is huge??? (and many queries)

KD-Trees

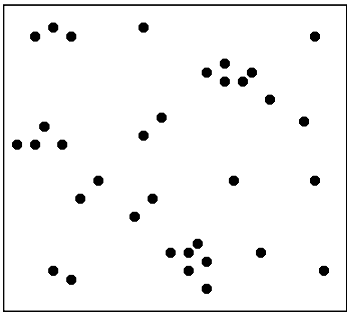
- Smarter approach: ***kd-trees***
  - Structured organization of documents
    - Recursively partitions points into axis aligned boxes.
  - Enables more efficient pruning of search space
    - Examine nearby points first.
    - Ignore any points that are further than the nearest point found so far.
- ***kd-trees*** work "well" in "low-medium" dimensions d
  - We'll get back to this…





KD-Tree Construction

2d example

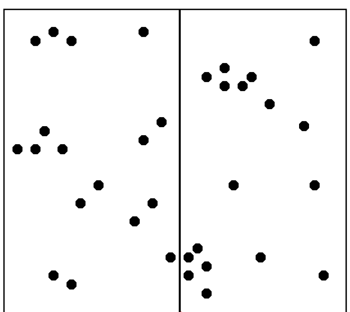


Pt	X	Y
1	0.00	0.00
2	1.00	4.31
3	0.13	2.85
…	…	…

x^1
x^2
x^3
$\vdots$

- Start with a list of *d*-dimensional points.

KD-Tree Construction



$x \le V$ V $x > V$

X>.5

NO

Pt	X	Y
1	0.00	0.00
3	0.13	2.85
…	…	…

YES

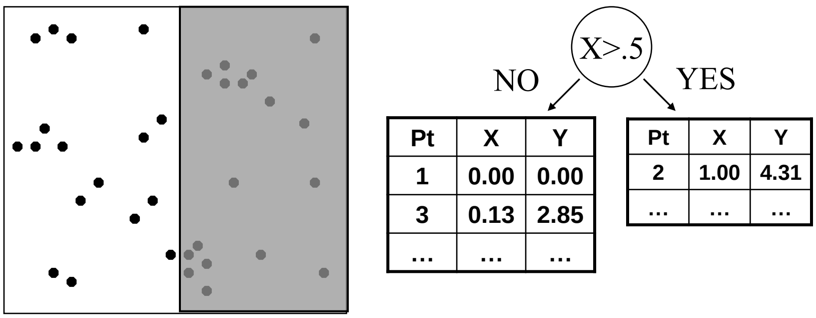
Pt	X	Y
2	1.00	4.31
…	…	…

X in this case

in this case V=0.5

- Split the points into 2 groups by:
  - Choosing dimension d_j and value V (methods to be discussed…)
  - Separating the points into $x^i_{d_j}$ > V and $x^i_{d_j}$ <= V.

KD-Tree Construction



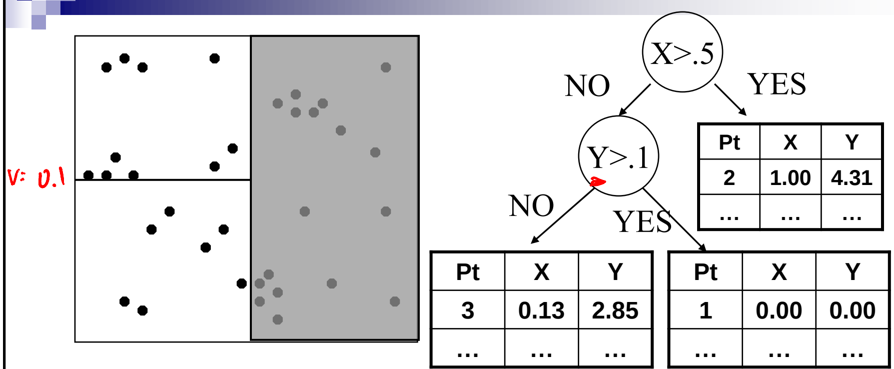
Pt	X	Y
1	0.00	0.00
3	0.13	2.85
…	…	…

Pt	X	Y
2	1.00	4.31
…	…	…

- Consider each group separately and possibly split again (along same/different dimension).
  - Stopping criterion to be discussed…

D how to choose dimension. .

KD-Tree Construction



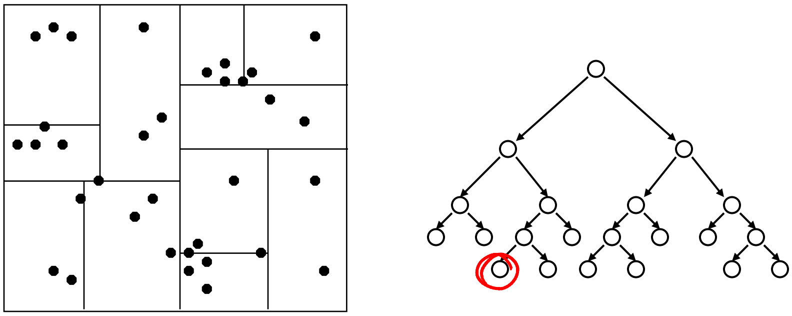
Pt	X	Y
3	0.13	2.85
…	…	…

Pt	X	Y
1	0.00	0.00
…	…	…

Pt	X	Y
2	1.00	4.31
…	…	…

- Consider each group separately and possibly split again (along same/different dimension).
  - Stopping criterion to be discussed…

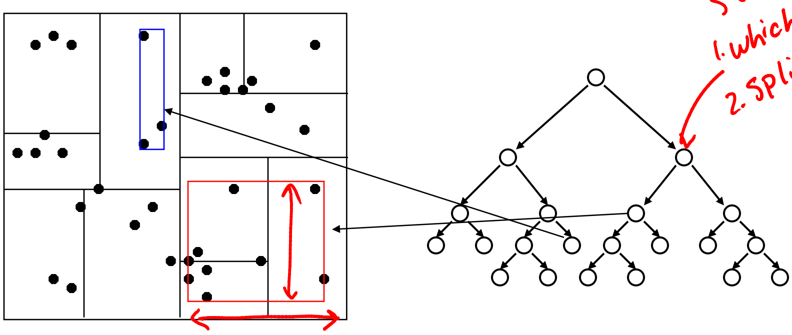
KD-Tree Construction



- Continue splitting points in each set
  - creates a binary tree structure
- Each leaf node contains a list of points

satisfying all conditions down the tree to that point

KD-Tree Construction



Store:
1. which dim?
2. split value

3. Keep one additional piece of information at each node:
   - The (tight) bounds of the points at or below this node.

KD-Tree Construction

Use heuristics to make splitting decisions:

- Which dimension do we split along?

  widest (or alternate)

- Which value do we split at?

  median of chosen split dim (or center)

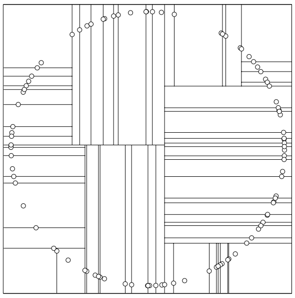
- When do we stop?

  fewer than m pts left

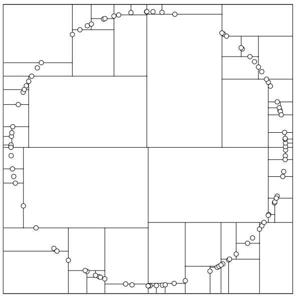
  OR

  box hits minimum width

Many heuristics…

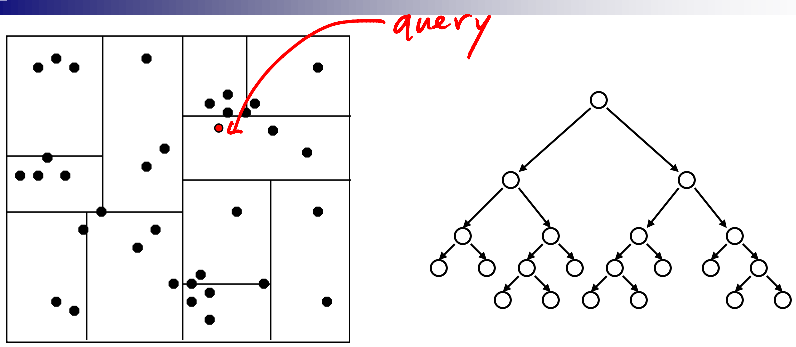


median heuristic



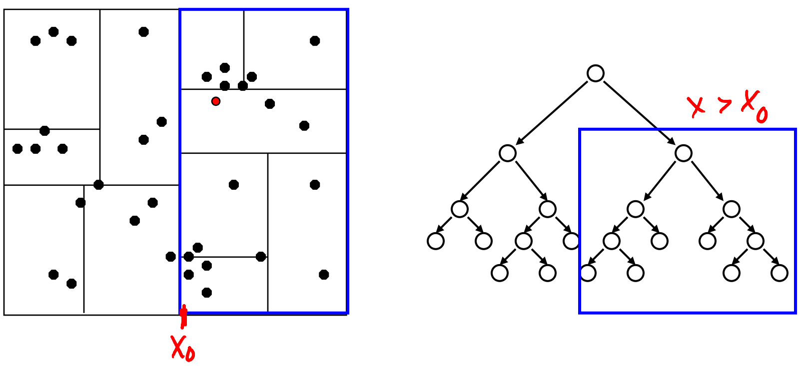
center-of-range heuristic

Nearest Neighbor with KD Trees



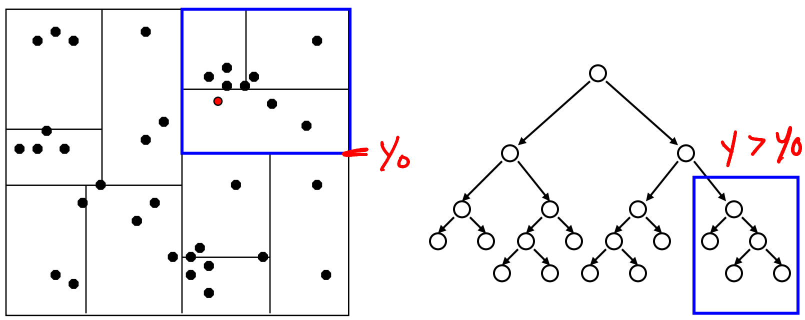
- Traverse the tree looking for the nearest neighbor of the query point.

Nearest Neighbor with KD Trees



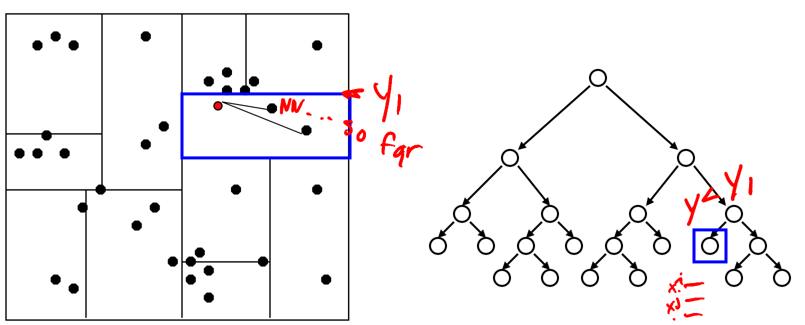
- <u>Examine nearby points first</u>:
  - Explore branch of tree closest to the query point first.

Nearest Neighbor with KD Trees



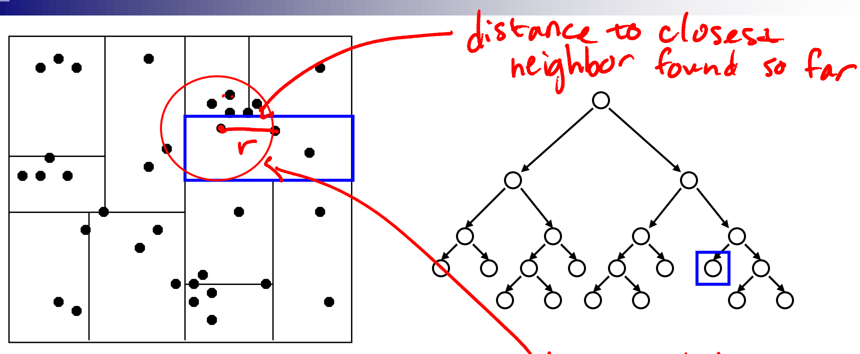
- <u>Examine nearby points first</u>:
  - Explore branch of tree closest to the query point first.

Nearest Neighbor with KD Trees



- When we reach a leaf node:
  - Compute the distance to each point in the node.

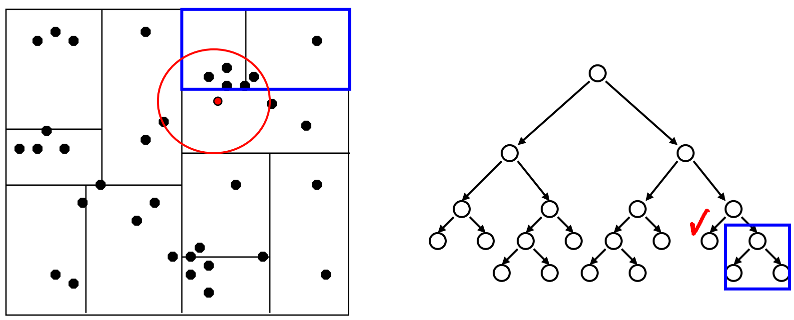
Nearest Neighbor with KD Trees



does NN have to be in this box?
NO

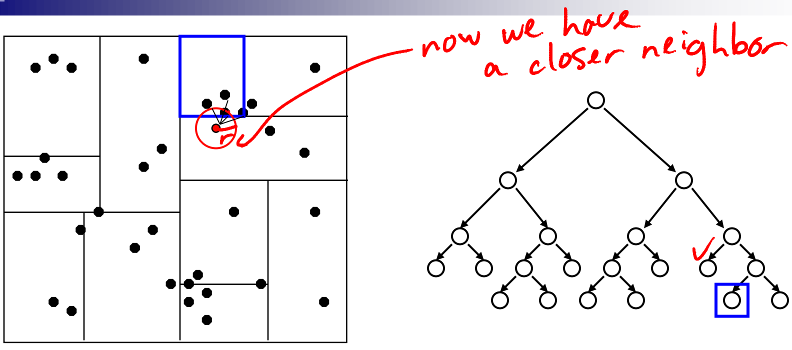
- When we reach a leaf node:
  - Compute the distance to each point in the node.

Nearest Neighbor with KD Trees



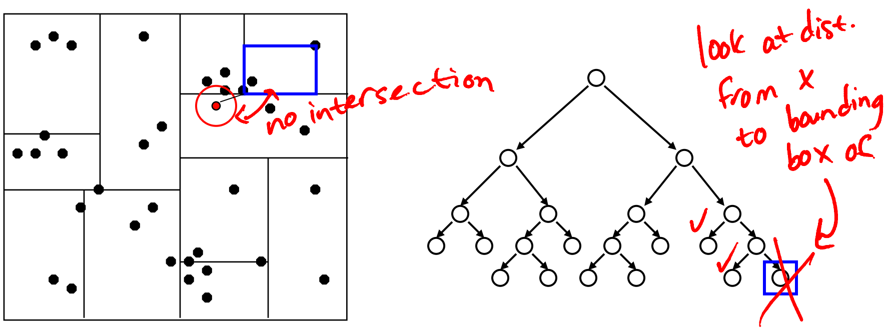
- Then backtrack and try the other branch at each node visited

Nearest Neighbor with KD Trees



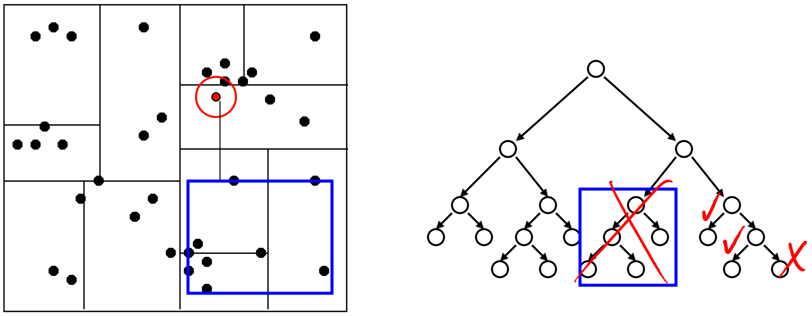
- Each time a new closest node is found, update the distance bound

Nearest Neighbor with KD Trees



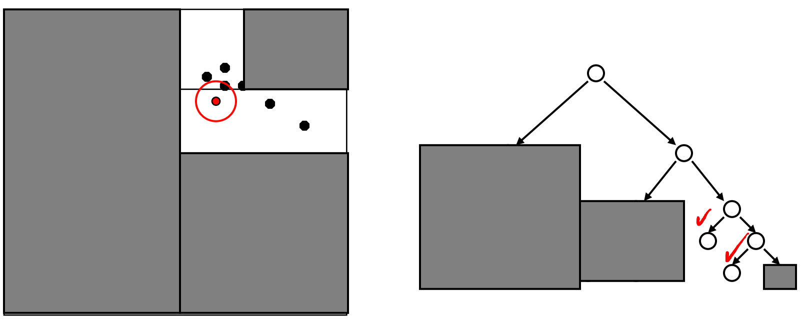
- Using the distance bound and bounding box of each node:
  - Prune parts of the tree that could NOT include the nearest neighbor

Nearest Neighbor with KD Trees



- Using the distance bound and bounding box of each node:
  - Prune parts of the tree that could NOT include the nearest neighbor

Nearest Neighbor with KD Trees

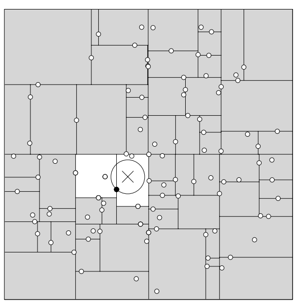


- Using the distance bound and bounding box of each node:
  - Prune parts of the tree that could NOT include the nearest neighbor

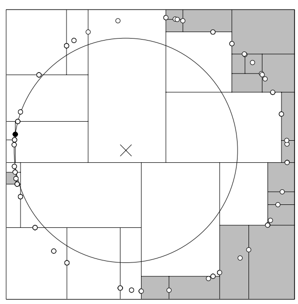
Complexity

- For (nearly) balanced, binary trees...
- Construction
  - Size: $2N-1 \rightarrow O(N)$
  - Depth: $O(\log N)$
  - Median + send points left right: $O(N)$ at every tree level (smart)
  - Construction time: $O(N \log N)$
- 1-NN query
  - Traverse down tree to starting point: $O(\log N)$
  - Maximum backtrack and traverse: $O(N)$ worst case
  - Complexity range: $O(\log N) \rightarrow O(N)$
- Under some assumptions on distribution of points, we get O(log*N*) but exponential in *d* (see citations in reading)

Complexity



pruned many (closer to $O(\log N)$)



pruned only a few $O(N)$

Complexity for N Queries

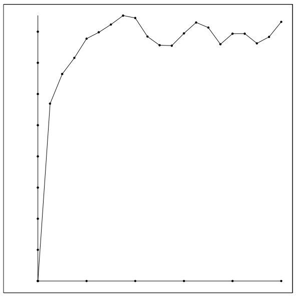
- Ask for nearest neighbor to each document

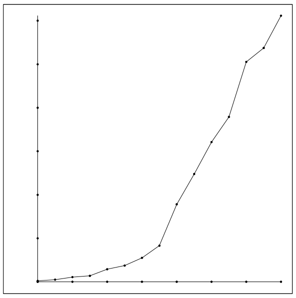
  N queries

- Brute force 1-NN: $O(N^2)$
- kd-trees: $O(N \log N) \rightarrow O(N^2)$

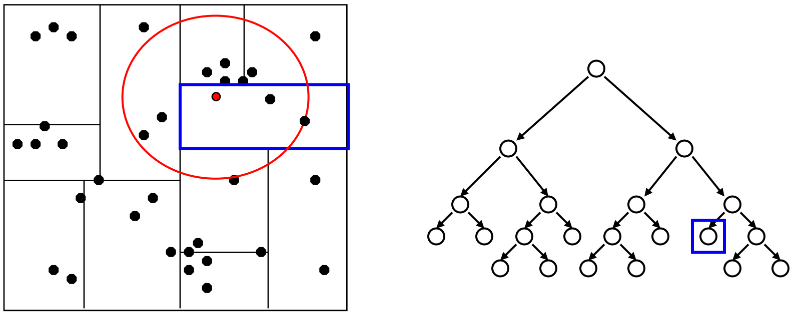
  potentially large savings!

Inspections vs. *N* and *d*



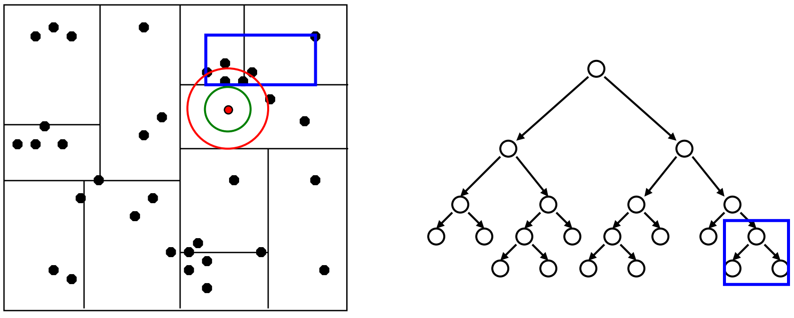


K-NN with KD Trees



- Exactly the same algorithm, but maintain distance as distance to furthest of current *k* nearest neighbors
- Complexity is:

Approximate K-NN with KD Trees



- **Before:** Prune when distance to bounding box >
- **Now:** Prune when distance to bounding box >
- Will prune more than allowed, but can guarantee that if we return a neighbor at distance r, then there is no neighbor closer than r/α.
- In practice this bound is loose…Can be closer to optimal.
- Saves lots of search time at little cost in quality of nearest neighbor.

Wrapping Up – Important Points

kd-trees

- Tons of variants
  - On construction of trees (heuristics for splitting, stopping, representing branches…)
  - Other representational data structures for fast NN search (e.g., ball trees,…)

Nearest Neighbor Search

- Distance metric and data representation are crucial to answer returned

For both…

- High dimensional spaces are hard!
  - Number of kd-tree searches can be exponential in dimension
    - Rule of thumb… $N >> 2^d$… Typically useless.
  - Distances are sensitive to irrelevant features
    - Most dimensions are just noise → Everything equidistant (i.e., everything is far away)
    - Need technique to learn what features are important for your task

What you need to know

- Document retrieval task
  - Document representation (bag of words)
  - tf-idf
- Nearest neighbor search
  - Formulation
  - Different distance metrics and sensitivity to choice
  - Challenges with large *N*
- kd-trees for nearest neighbor search
  - Construction of tree
  - NN search algorithm using tree
  - Complexity of construction and query
  - Challenges with large *d*

Acknowledgment

- This lecture contains some material from Andrew Moore's excellent collection of ML tutorials:
  - http://www.cs.cmu.edu/~awm/tutorials
- In particular, see:
  - http://grist.caltech.edu/sc4devo/.../files/sc4devo_scalable_datamining.ppt