Case Study 1: Estimating Click Probabilities

Tackling an Unknown Number of Features with Sketching

Machine Learning for Big Data
CSE547/STAT548, University of Washington
Emily Fox
April 9th, 2015

Problem 1: Complexity of LR Updates

- Logistic regression update: stochastic gradient ascent

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta_t \left\{ -\lambda w_i^{(t)} + x_i^{(t)} [y^{(t)} - P(Y=1|\mathbf{x}^{(t)}, \mathbf{w}^{(t)})] \right\}$$

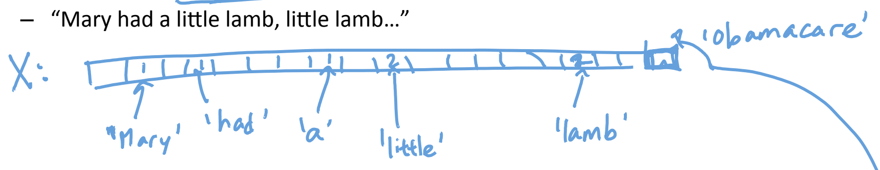
- Complexity of updates:
  - Constant in number of data points ✓
  - In number of features? O(d)
    - Problem both in terms of computational complexity and sample complexity

what if we have 1B features? large d

- What can we with very high dimensional feature spaces?
  - Kernels not always appropriate, or scalable "kernel trick"
  - What else?

Problem 2: Unknown Number of Features

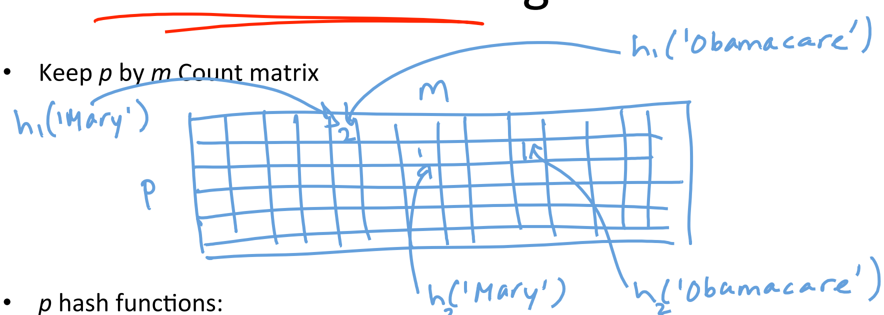
- For example, bag-of-words features for text data:
  - "Mary had a little lamb, little lamb…"



- What's the dimensionality of **x**? size of vocabulary… millions
- What if we see new word that was not in our vocabulary? growing d
  - Obamacare
  - Theoretically, just keep going in your learning, and initialize $\mathbf{w}_{Obamacare} = 0$
  - In practice, need to re-allocate memory, fix indices,… A big problem for Big Data

Count-Min Sketch: general case

- Keep *p* by *m* Count matrix



- *p* hash functions:
  - Just like in Bloom Filter, decrease errors with multiple hashes
  - Every time see string *i*:

j^th hash fcn

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

row input

Querying the Count-Min Sketch

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

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

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

  - Thus:

Q(i)

each $Count[j, h(i)] \geq a_i$

  - Return:

estimate

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

← tightest upper bound

©Emily Fox 2015

But updates may be positive or negative

$$w_i^{(t+1)} \leftarrow \overbrace{w_i^{(t)} + \eta_t \left\{-\lambda w_i^{(t)} + x_i^{(t)} \underbrace{[y^{(t)} - P(Y=1|\mathbf{x}^{(t)}, \mathbf{w}^{(t)})]}_{\text{pos. or neg., non-integer}}\right\}}^{(1-\eta_t\lambda) w_i^{(t)} + \eta_t x_i^{(t)} (y^{(t)} - P(Y=1|\ldots))}$$

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

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

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

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

count[1, $h_1(i)$]

count[3, $h_3(i)$]

a_i →

count[2, $h_2(i)$]

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

©Emily Fox 2015

Finally, Sketching for LR

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta_t \left\{ -\lambda w_i^{(t)} + x_i^{(t)} [y^{(t)} - P(Y = 1 | \mathbf{x}^{(t)}, \mathbf{w}^{(t)})] \right\}$$

- Never need to know size of vocabulary!
- At every iteration, update Count-Min matrix:

$\forall j,k \quad count[j,k] = (1-\eta_t\lambda)\, count[j,k]$

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

$\quad \forall j \quad count[j, h_j(i)] \mathrel{+}= x_i^{(t)} \cdot const$

$\quad\quad const = \eta_t (y^{(t)} - P(Y=1 \ldots))$

- Making a prediction:

Remember our est. of $w_i^{(t)}$: $\text{median}_j \; count[j, h_j(i)]$

Make pred:

$-\log odds = w_0^{(t)} + \sum_{i: x_i^{(t)} \neq 0} \left(\text{median}_j \; count[j, h_j(i)]\right) \cdot x_i^{(t)}$

- Scales to huge problems, great practical implications…

Hash Kernels

- Count-Min sketch not designed for negative updates
- Biased estimates of dot products
- $\boxed{\textbf{Hash Kernels}}$: Very simple, but powerful idea to remove bias
- Pick 2 hash functions:
  - h : Just like in Count-Min hashing $\quad h : X \rightarrow \{1, \ldots, m\}$
  - ξ : Sign hash function $\quad \xi : X \rightarrow \{+1, -1\}$
    - Removes the bias found in Count-Min hashing (see homework)
- Define a "kernel", a projection ϕ for $\mathbf{x}$:

$\phi(x) =$ [| | | | | | | | | | | | | |]

For each non-zero element of x, add to bin $h(j)$: $\xi(j) x_j$

$$\phi_i(x) = \sum_{j : h(j) = i} \xi(j)\, x_j$$

If $x_j = 7$

$h(j) = 4$

$\xi(j) = -1$

$\Downarrow$

add -7 to bin 4

can think of as a random projection of x

Hash Kernels Preserve Dot Products

$$\phi_i(\mathbf{x}) = \sum_{j:h(j)=i} \xi(j)\mathbf{x}_j$$

- Hash kernels provide unbiased estimate of dot-products!

$E_{h,\xi}[\phi(x)\cdot\phi(y)] = x\cdot y$ Pf: homework :)

- Variance decreases as O(1/*m*) ← gets better w/ more dims

- Choosing *m*? For ε>0, if

$$m = \mathcal{O}\left(\frac{\log\frac{N}{\delta}}{\epsilon^2}\right)$$

log in data size

  - Under certain conditions…
  - Then, with probability at least 1-δ: high prob.

$$(1-\epsilon)||\mathbf{x}-\mathbf{x}'||_2^2 \leq ||\phi(\mathbf{x})-\phi(\mathbf{x}')||_2^2 \leq (1+\epsilon)||\mathbf{x}-\mathbf{x}'||_2^2$$

Learning With Hash Kernels

- Given hash kernel of dimension *m*, specified by *h* and *ξ*
  - Learn *m* dimensional weight vector
- Observe data point **x**
  - Dimension does not need to be specified a priori!
- Compute *ϕ*(**x**):
  - Initialize *ϕ*(**x**) = 0
  - For non-zero entries *j* of $\mathbf{x}_j$:

$\phi_{h(j)} \mathrel{+}= \xi(j)\, x_j$

e.g. j='Mary' h('Mary')=7 ξ('Mary')=-1

$\phi_7 \mathrel{+}= -x_{\text{'Mary'}}$

- Use normal update as if observation were *ϕ*(**x**), e.g., for LR using SGD:

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta_t\left\{-\lambda w_i^{(t)} + \phi_i(\mathbf{x}^{(t)})[y^{(t)} - P(Y=1|\phi(\mathbf{x}^{(t)}), \mathbf{w}^{(t)})]\right\}$$

↑ w length m, i = 0, 1, …, m-1

$$P(Y=1\mid\phi(x^{(t)}), w^{(t)}) = \frac{\exp(\phi(x^{(t)})\cdot w^{(t)})}{1+\exp(\phi(x^{(t)})\cdot w^{(t)})}$$

Interesting Application of Hash Kernels: Multi-Task Learning

- Personalized click estimation for many users:
  - One global click prediction vector **w**: predict using $w \cdot x$ ← $\frac{\exp(w \cdot x)}{1+\exp(w \cdot x)}$
    - But… people are unique
  - A click prediction vector $\mathbf{w}_u$ per user u: predict with $w_u \cdot x$
    - But… people don't each provide much data (labels)
- Multi-task learning: Simultaneously solve multiple learning related problems:
  - Use information from one learning problem to inform the others
- In our simple example, learn both a global **w** and one $\mathbf{w}_u$ per user: now represents deviation from global
  - Prediction for user u: $(w+w_u)\cdot x = w \cdot x + w_u \cdot x$
  - If we know little about user u: basically, $w \cdot x$
  - After a lot of data from user u: using $w + w_u$ as your vector

Problems with Simple Multi-Task Learning

- Dealing with new user is annoying, just like dealing with new words in vocabulary
- Dimensionality of joint parameter space is HUGE, e.g. personalized email spam classification from Weinberger et al.:
  - 3.2M emails
  - 40M unique tokens in vocabulary
  - 430K users
  - 16T parameters needed for personalized classification!

Hash Kernels for Multi-Task Learning

- Simple, pretty solution with hash kernels:
  - Very multi-task learning as (sparse) learning problem with (huge) joint data point **z** for point **x** and user *u*:

$$z_{(x,u)} = (\underbrace{x_1,\dots,x_d}_{\text{global}}, \overbrace{\underbrace{0,\dots,0}_{d}, \dots, \underbrace{x_1,\dots,x_d}_{\text{user } u}, 0,\dots,0}^{\#\text{ of users}})$$

- Estimating click probability as desired:

$$\underline{w} = (\underset{\text{global}}{w}, w_1, \dots, w_u, \dots, w_{\#\text{ users}})$$

$$z_{(x,u)} \cdot \underline{w} = w \cdot x + w_u \cdot x$$

- Address huge dimensionality, new words, and new users using hash kernels:

$$\phi(z_{(x,u)}): \qquad \phi_i = \sum_{j:h(j)=i} \xi(j)\, z_{(x,u),j}$$

Simple Trick for Forming Projection $\phi(\mathbf{x},u)$

In practice, don't form + then project $z_{(x,u)}$

- Observe data point **x** for user *u*
  - Dimension does not need to be specified a priori and user can be new!
- Compute $\phi(\mathbf{x},u)$:
  - Initialize $\phi(\mathbf{x},u) = 0$
  - For non-zero entries *j* of $\mathbf{x}_j$:
    - E.g., j='Obamacare'
    - Need two contributions to ϕ:
      - Global contribution $\quad \phi_{h(\text{'obamacare'})} \mathrel{+}= \xi(\text{'obamacare'})\, x_j$
      - Personalized Contribution $\quad$ augment vocabulary:
    - Simply: $u = 17$ $\quad$ index specific to user 17

$$\phi_{h(\text{'obamacare\_user17'})} \mathrel{+}= \xi(\text{'obamacare\_user17'}) \cdot x_j$$

- Learn as usual using $\phi(\mathbf{x},u)$ instead of $\phi(\mathbf{x})$ in update function

Results from Weinberger et al. on Spam Classification: Effect of *m*

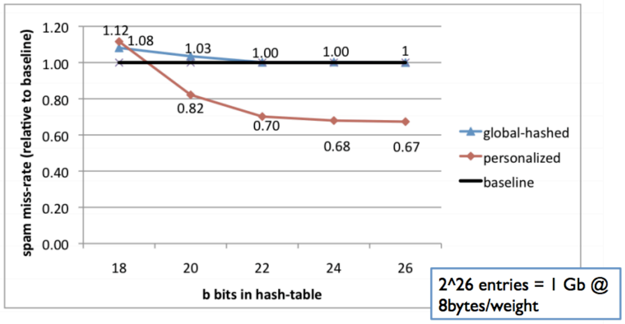


Figure 2. The decrease of uncaught spam over the baseline classifier averaged over all users. The classification threshold was chosen to keep the not-spam misclassification fixed at 1%. The hashed global classifier (*global-hashed*) converges relatively soon, showing that the distortion error ϵ_d vanishes. The personalized classifier results in an average improvement of up to 30%.

 15

Results from Weinberger et al. on Spam Classification: Multi-Task Effect

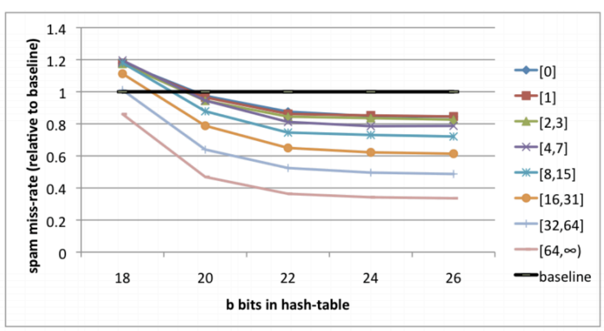


Figure 3. Results for users clustered by training emails. For example, the bucket $[8, 15]$ consists of all users with eight to fifteen training emails. Although users in buckets with large amounts of training data do benefit more from the personalized classifier (up-to 65% reduction in spam), even users that did not contribute to the training corpus at all obtain almost 20% spam-reduction.

 16

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 logistic regression
- Hash kernels:
  - Sparse representation for feature vectors
  - Very simple, use two hash function (Can use one hash function…take least significant bit to define ξ)
  - Quickly generate projection $\varphi(\mathbf{x})$
  - Learn in projected space
- Multi-task learning:
  - Solve many related learning problems simultaneously
  - Very easy to implement with hash kernels
  - Significantly improve accuracy in some problems (if there is enough data from individual users)

Case Study 2: Document Retrieval

Task Description: Finding Similar Documents

Machine Learning for Big Data
CSE547/STAT548, University of Washington
Emily Fox
April 9th, 2015

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 counts

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

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

1-Nearest Neighbor

- Articles $\mathcal{X} = \{x^1, \dots, x^N\}, \quad x^i \in \mathbb{R}^d$
- Query: x
- 1-NN
  - Goal: find article in $\mathcal{X}$ "closest" to x

    ☆ need distance metric ☆ $d(u,v)$
  - Formulation:

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

k-Nearest Neighbor

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

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

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

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

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 dim i

Equivalently,

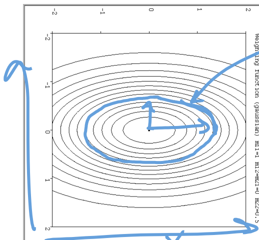
$$d(u,v) = \sqrt{(u-v)'\Sigma(u-v)}$$

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)



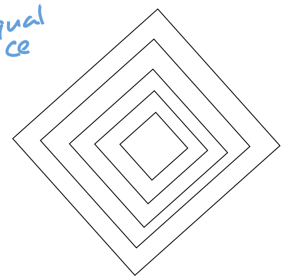
Scaled Euclidian (L_2)

x_2

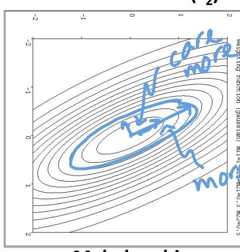
x_1

level set = pts of equal distance

care more about changes in x_2



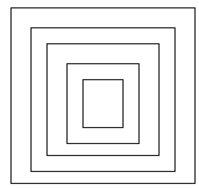
L_1 norm (absolute)



care more

more variability $\Rightarrow$ care less

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



L_∞ *(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? replicate α times
  - 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

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

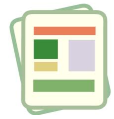
Issues with Document Representation

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

rare words swamped by common words

"The koala hugs the tree..." ⟷ "Tree huggers..."

common words like "tree" and "hug" are dominating



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

TF-IDF

- Term frequency:

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

term, doc

$$\frac{f(t,d)}{\max\{f(w,d);\, w \in d\}}$$

prevents bias towards long doc

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

- Inverse document frequency:

$$\mathrm{idf}(t, \cancel{D}\,\mathcal{X}) = \log \frac{|\mathcal{X}|}{1+|\{d \in \mathcal{X} : t \in d\}|}$$

$\to 0$ $t \in$ many docs

> 0 o.w.

- tf-idf:

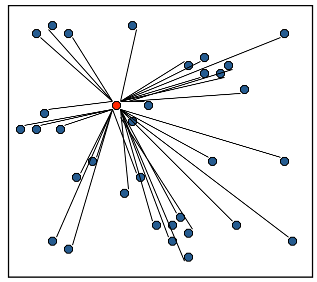
$$\mathrm{tfidf}(t,d,\cancel{D}\,\mathcal{X}) = \mathrm{tf}(t,d) \times \mathrm{idf}(t,\mathcal{X})$$

  - 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!

↖ keep priority queue of top k
+ inserting into queue is log k

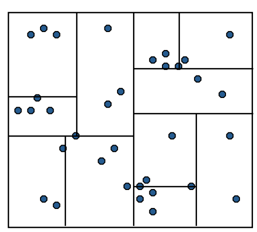


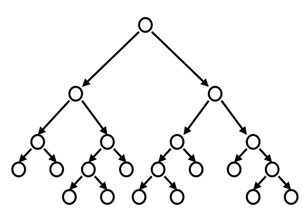
33 Distance Computations

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

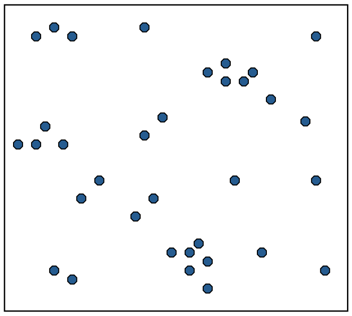
KD-Trees

- 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

word 1   word 2

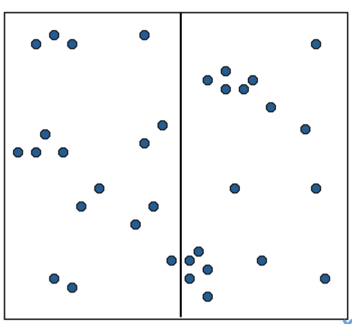
Pt	X	Y
1	0.00	0.00
2	1.00	4.31
3	0.13	2.85
…	…	…

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

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

KD-Tree Construction

y



$x \le v$ v $x > v$ x

v

X>.5

NO   YES

Pt	X	Y
1	0.00	0.00
3	0.13	2.85
…	…	…

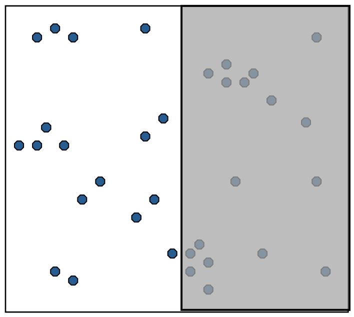
Pt	X	Y
2	1.00	4.31
…	…	…

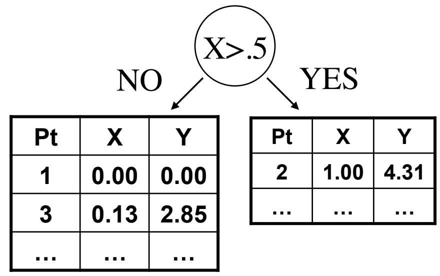
x in this case

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





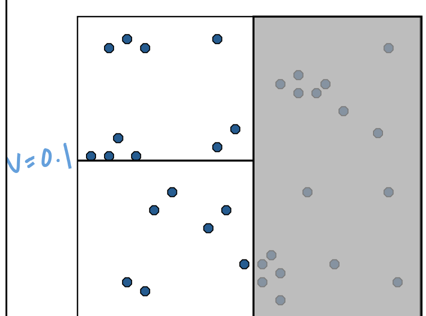
Pt	X	Y
1	0.00	0.00
3	0.13	2.85
…	…	…

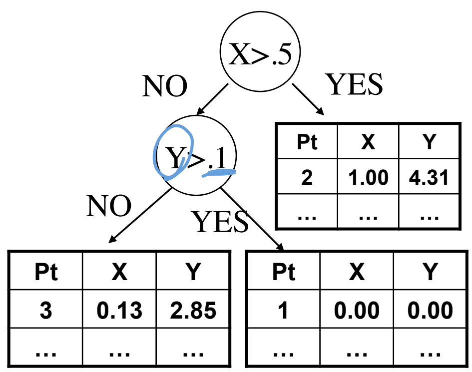
Pt	X	Y
2	1.00	4.31
…	…	…

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

 33

KD-Tree Construction





Pt	X	Y
2	1.00	4.31
…	…	…

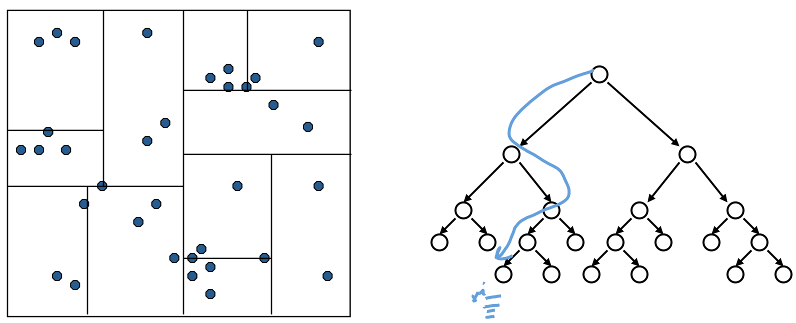
Pt	X	Y
3	0.13	2.85
…	…	…

Pt	X	Y
1	0.00	0.00
…	…	…

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

 34

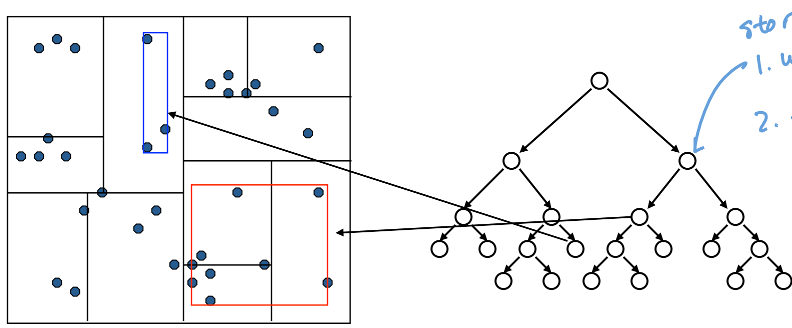
KD-Tree Construction



- 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.

