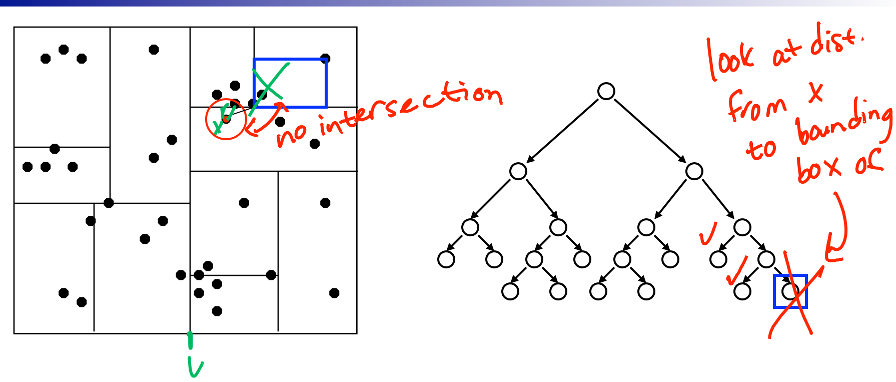
Case Study 2: Document Retrieval

Finding Similar Documents Using Nearest Neighbors

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

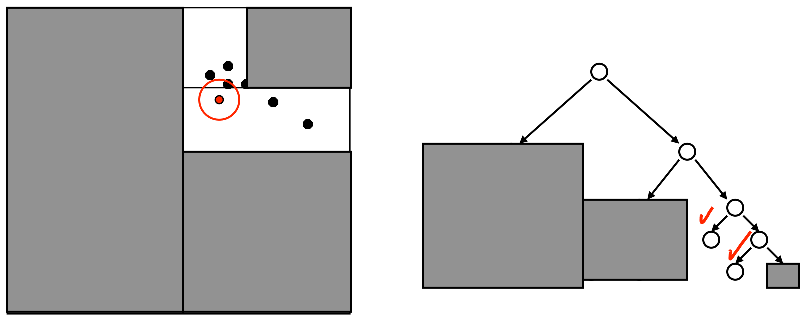
January 22nd, 2013

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



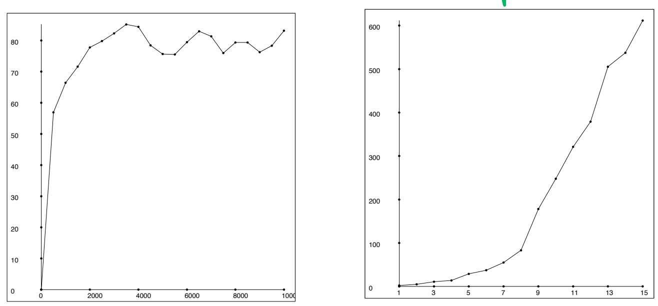
- Using the distance bound and bounding box of each node:
  - Prune parts of the tree that could NOT include the nearest neighbor

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
  - Construction time: $O(N \log N) \leftarrow$ (smart)
- 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)

Inspections vs. *N* and *d*

log N   exponential



\# of inspections

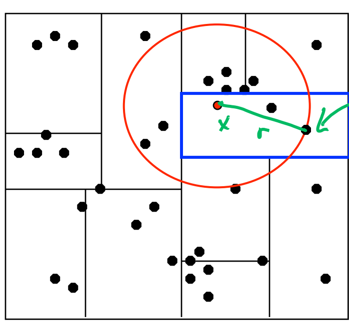
N   d

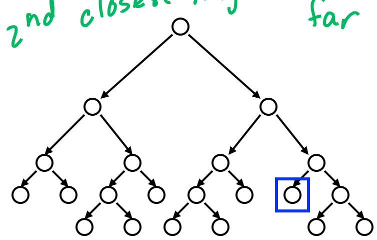
$(x^i \in \mathbb{R}^d)$

K-NN with KD Trees

2-NN example

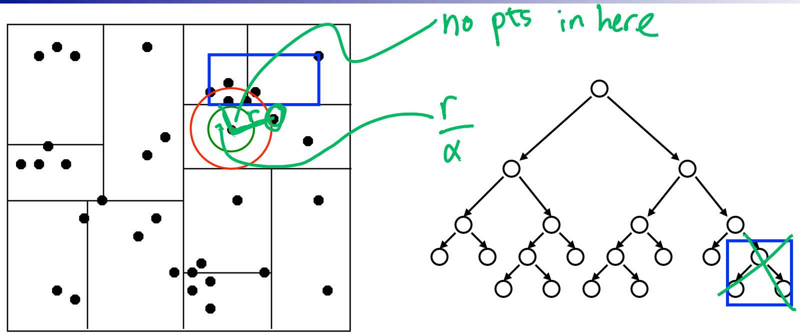
2nd closest neighbor found so far





- Exactly the same algorithm, but maintain distance as distance to furthest of current *k* nearest neighbors
- Complexity is: $O(k \log N)$

Approximate K-NN with KD Trees



$\alpha > 1$

- **Before:** Prune when distance to bounding box > r
- **Now:** Prune when distance to bounding box > r/α
- 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! large d
  - Number of kd-tree searches can be exponential in dimension
    - Rule of thumb… $N \gg 2^d$… 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

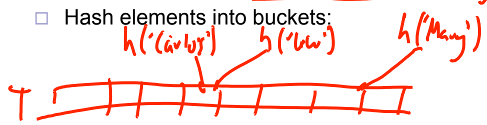
- Document retrieval task
  - Document representation (bag of words)
  - tf-idf
- Nearest neighbor search
  - Formulation
  - Different distance metrics and sensitivity to choice
  - Challenges with large *N*
- kd-trees for nearest neighbor search
  - Construction of tree
  - NN search algorithm using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

Locality-Sensitive Hashing
Hash Kernels
Multi-task Learning

Machine Learning/Statistics for Big Data
CSE599C1/STAT592, University of Washington
Carlos Guestrin
January 24th, 2013

Using Hashing to Find Neighbors

- KD-trees are cool, but…
  - Non-trivial to implement efficiently
  - Problems with high-dimensional data
- Approximate neighbor finding…
  - Don't find exact neighbor, but that's OK for many apps, especially with Big Data
- What if we could use hash functions: $h: X \to \{1, \ldots, m\}$
  - Hash elements into buckets:



  - Look for neighbors that fall in same bucket as **x**:

$h(x) = i$, for all $y \in T[h(x) = i]$ look for neighbors there

- But, by design…

by design $P(h(x) = h(x')) = \frac{1}{m} \quad \forall x'$

even if $d(x, x')$ is low $\not\Rightarrow$ $h(x) = h(x')$

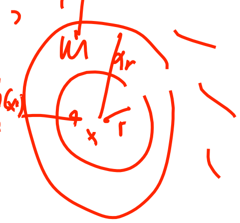
Locality Sensitive Hashing (LSH)

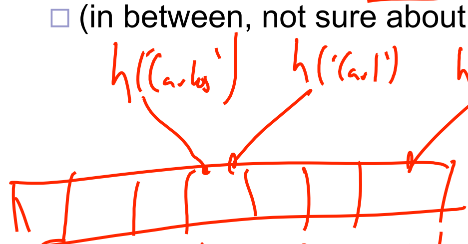
don't know

- A LSH function *h* satisfies (for example), for some some similarity function *d*, for r>0, α>1:
  - d(**x**,**x**') ≤ r, then P(h(**x**)=h(**x**')) is high
  - d(**x**,**x**') > α.r, then P(h(**x**)=h(**x**')) is low
  - (in between, not sure about probability)

$h(x) \neq h(x)$

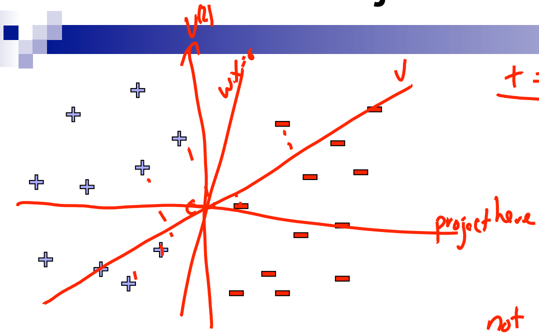
$h(x) = h(x)$ prob





now we look for points hashing to same bin

Random Projection Illustration



$+ + + + + + \;\vdots\; - - - - - -$

project on v: ← good

$+ + + + + - + - - \; - \; -$

$v x$ ← little overlap but mostly separated

not as good → $v^{(2)}$

$- + - + - + - + +$

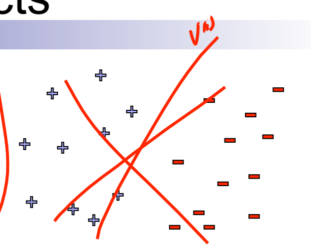
$v^{(2)} \cdot x$

- Pick a random vector **v**:
  - Independent Gaussian coordinates

  $v_i \overset{iid}{\sim} N(0,1)$
- Preserves separability for most vectors
  - Gets better with more random vectors

©Carlos Guestrin 2013 | 13

Multiple Random Projections: Approximating Dot Products



- Pick m random vectors $\mathbf{v}^{(i)}$:
  - Independent Gaussian coordinates of projection vectors $v_j^{(i)} \overset{iid}{\sim} N(0,1)$

$$\phi(x) = \begin{pmatrix} v^{(1)} \cdot x \\ v^{(2)} \cdot x \\ \vdots \\ v^{(m)} \cdot x \end{pmatrix}$$

- Approximate dot products:
  - Cheaper, e.g., learn is smaller m dimensional space

$$x \cdot y \approx \frac{1}{m} \phi(x) \cdot \phi(y) = \frac{1}{m} \sum_{i=1}^{m} (v^{(i)} \cdot x)(v^{(i)} \cdot y)$$

- Only need logarithmic number of dimensions!
  - N data points, approximate dot-product within $\varepsilon>0$:

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

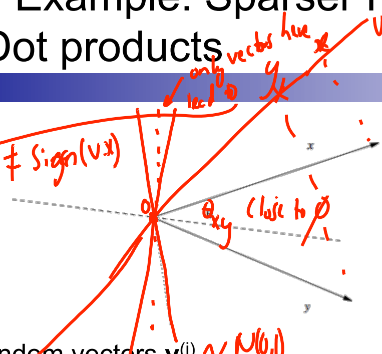
if I have big data, $N \to$ very large, but only need $\log N$ random vectors

$v^{(i)}$ are dense w prob 1 } even if x is sparse
$\Rightarrow v^{(i)} \cdot x \neq 0$ w. prob 1 } $\phi(x)$ is not sparse

- But all sparsity is lost

©Carlos Guestrin 2013 | 14

LSH Example: Sparser Random Projection for Dot products



with high prob. $\mathrm{sign}(y \cdot v) = \mathrm{sign}(x \cdot v)$

- Pick random vectors $\mathbf{v}^{(i)} \sim N(0,1)$
- Simple 0/1 projection: $h_i(x) = \begin{cases} 1 & \text{if } \mathrm{sign}(v^{(i)} \cdot x) \geq 0 \\ 0 & \text{if } \mathrm{sign}(v^{(i)} \cdot x) < 0 \end{cases}$
- Now, each vector is approximated by a bit-vector
  $\phi(x) = (0,0,1,0,1,1,1,0)$
- Dot-product approximation:

$$\frac{x \cdot y}{\|x\|\,\|y\|} = \cos \theta_{xy} \approx 1 - 2\frac{\text{HammingDistance}(\phi(x), \phi(y))}{m} = 1 - 2\frac{\|\phi(x) - \phi(y)\|_2^2}{m}$$

LSH for Approximate Neighbor Finding

- Very similar elements fall in exactly same bin: $d(x,y) = \cos\theta_{xy}$ (similarity) (cosine similarity) $\approx 1 - \frac{\text{hamming}(\phi(x),\phi(y))}{m}$

  $x \rightarrow T[\phi(x)]$ turn bit-vector into int

  T [| | | | | | | | |] ← 2^m bins

  $d(x,y)$ = high $\Rightarrow \cos\theta_{xy} \approx 1 \Rightarrow$ hamming distance $(\phi(x), \phi(y)) \approx 0 \Rightarrow x, y$ same bin
- And, nearby bins are also nearby:
  in terms of hamming distance
- Simple neighbor finding with LSH: for x
  - For bins b of increasing hamming distance to $h(\mathbf{x})$:
    - Look for neighbors of **x** in bin b
      $\forall y$ in $T[b]$, compute $d(x,y)$, keep closest
  - Stop when run out of time
- Pick m such that $N/2^m$ is "smallish" (in practice)

Hash Kernels: Even Sparser LSH for Learning

- Two big problems with random projections:
  - Data is sparse, but random projection can be a lot less sparse
  - You have to sample *m* huge random projection vectors
    - And, we still have the problem with new dimensions, e.g., new words
- **Hash Kernels**: Very simple, but powerful idea: combine sketching for learning with random projections
- Pick 2 hash functions:
  - *h* : Just like in Min-Count hashing
  - ξ : Sign hash function
    - Removes the bias found in Min-Count hashing (see homework)
- Define a "kernel", a projection ϕ for **x**:

$v^{(i)}$ dense vectors with 40M entries

$h: X \to \{1,\ldots,m\}$

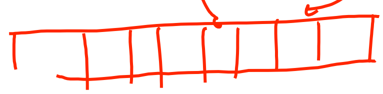
$\xi: X \to \{+1,-1\}$

$\phi(x) =$

for each non-zero element j of x add to bin $h(j)$ $\xi(j)\cdot x_j$

if $x_j = 7$, $h(j)=4$, $\xi(j)=-1$: add -7 to bin 4

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

$\phi(x) =$ 

©Carlos Guestrin 2013

Hash Kernels, Random Projections and Sparsity

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

$h(\text{'a'})=7$ $\xi(\text{'a'})=-1$

$h(\text{'lamb'})=7$ $\xi(\text{'lamb'})=+1$

- Hash Kernel as a random projection:

$x = ($ 0 0 0 mary 1 0 0 had 1 a 1 0 0 little 2 0 0 lamb 2 $)$

$\phi(x) =$ 

$v^{(i)} = (0\ 0\ 0 +1\ 0\ 0\ 0 -1\ 0\ 0)$

- Random projection vector for coordinate *i* of ϕ_i:

$v^{(i)}$ mostly zero; non zero for $j: h(j)=i$; here $v_j^{(i)} = \{+1,-1\}$

- Implicitly define projection by *h* and ξ, so no need to compute a priory and automatically deal with new dimensions
- Sparsity of ϕ, if **x** has *s* non-zero coordinates: ? once! at $h(j)$

how many times does x_j "show up" in $\phi(x)$

thus if sparsity of $X = s \geq$ sparsity of $\phi(x)$

©Carlos Guestrin 2013

$||x-y||_2^2 = x^2 + y^2 - 2x \cdot y$

Hash Kernels Preserve Dot Products

- Hash kernels provide unbiased estimate of dot-products!

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

proof: by homework

- Variance decreases as O(1/*m*) ← gets better with 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:

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

Learning With Hash Kernels

- Given hash kernel of dimension *m*, specified by *h* and ξ
  - Learn *m* dimensional weight vector

$\phi(x) = \begin{pmatrix} 1 \\ \vdots \\ m \end{pmatrix}$

- 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., $j = \text{'uw'}$, $h(\text{'uw'}) = 7$, $\xi(\text{'uw'}) = -1$

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

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

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

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

Interesting Application of Hash Kernels: Multi-Task Learning

$$\frac{\exp(\mathbf{x}\cdot\mathbf{w})}{1+\exp(\mathbf{x}\cdot\mathbf{w})}$$

- Personalized click estimation for many users:
  - One global click prediction vector **w**: predict using $w \cdot x$
    - But… people are unique
  - A click prediction vector $\mathbf{w}_u$ per user *u*: predict with $w_u \cdot x$
    - But… people are lazy
- 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:
  - Prediction for user *u*: $(w + w_u)x = w \cdot x + w_u \cdot x$
  - 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 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*:

users dim: 16T

$$z_{(x,u)} = (x_1 \ldots x_n,\ 0\ 0\ 0\ 0\ 0,\ \underbrace{x_1 \ldots x_n}_{u},\ 0 \ldots)$$

- Estimating click probability as desired:

$$w = (w, w_1, \ldots, w_u, \ldots w_{\#users}) : \quad z_{(x,u)} \cdot w = w \cdot x + w_u \cdot x = (w + w_u) \cdot x$$

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

$\phi(z_{(x,u)})$ just like with hash kernel

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

  - Desired effect achieved if *j* includes both
    - just word (for global **w**)
    - word,user (for personalized $\mathbf{w}_u$)

j

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

- Observe data point **x** for user *u*
  - Dimension does not need to be specified a priori and user can be ~~unknown~~ 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 $\phi_{h(\text{'Obamacare'})} \mathrel{+}= \xi(\text{'Obamacare'})\, x_j$
      - Personalized Contribution
    - Simply: $u = 17$

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

- 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*

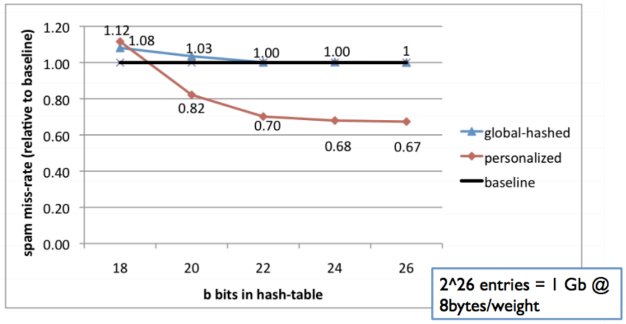


Figure 2. The decrease of uncaught spam over the baseline classifier averaged over all users. The classification threshold was chosen to keep the not-spam misclassification fixed at 1%. The hashed global classifier (*global-hashed*) converges relatively soon, showing that the distortion error ϵ_d vanishes.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

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

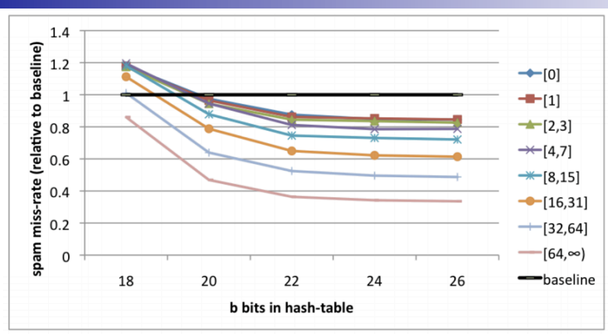


Figure 3. Results for users clustered by training emails. For example, the bucket $[8, 15]$ consists of all users with eight to fifteen training emails. Although users in buckets with large amounts of training data do benefit more from the personalized classifier (up-to 65% reduction in spam), even users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

What you need to know

- Locality-Sensitive Hashing (LSH): nearby points hash to the same or nearby bins
- LSH use random projections
  - Only O(log N/ε^2) vectors needed
  - But vectors and results are not sparse
- Use LSH for nearest neighbors by mapping elements into bins
  - Bin index is defined by bit vector from LSH
  - Find nearest neighbors by going through bins
- Hash kernels:
  - Sparse representation for feature vectors
  - Very simple, use two hash function
    - Can even use one hash function, and take least significant bit to define ξ
  - Quickly generate projection $\phi(\mathbf{x})$
  - 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