ECS289: Scalable Machine Learning

Cho-Jui Hsieh
UC Davis

Oct 25, 2016

Outline

- Nearest Neighbor Search
- Maximum Inner Product Search

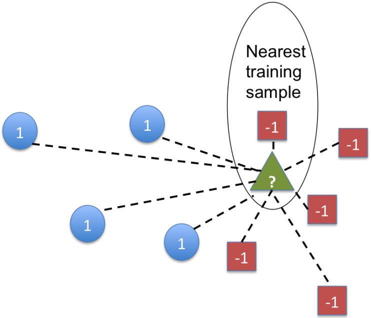
Nearest Neighbor Classification

- K-nearest neighbor classification

  Predict the label by voting among K-nearest neighbors.
- Prediction time: Find the nearest training sample

  1 billion samples, each distance evaluation requires 1 micro second

  $\Rightarrow$ 1000 secs per prediction



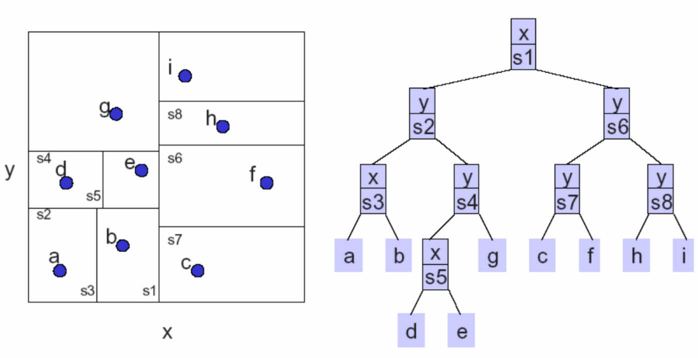
Nearest Neighbor Search

- Given a database with n vectors $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d\}$
- Given a query point $\boldsymbol{v}$, how to find the closest points in the database?
- Linear search:
  - Compute $\|\boldsymbol{x}_i - \boldsymbol{v}\|$ for all i
  - Need $O(nd)$ time
- Can we do better?

Tree-based Nearest Neighbor Search

Tree-based Approaches

- KD Tree (Bentley, 1975)
- Preprocessing:
  - Divide the points in half by a line perpendicular to one of the axes
  - Stop when there is only one or few points in a leaf



(Picture from

`http://andrewd.ces.clemson.edu/courses/cpsc805/references/nearest_search.pdf`)

KD-tree: Construction

- Sort the data points in each dimension
- Requires $O(dn)$ time to find the best split in each level

  (Scan over the sorted list for each dimension)
- Balanced construction: $O(\log n)$ levels
- Total time: $O(dn \log n)$

KD-tree: Query

- Recursively traverse down the tree.
- Assume the current nearest distance is r and

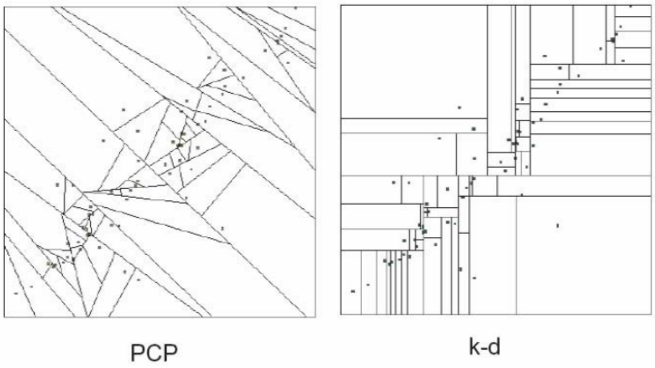
$$|\text{split\_value} - \text{x.split\_index}| < r$$

$\Rightarrow$ Only go down to the current branch

- Otherwise, need to check both branches.
- May need to check $O(n)$ nodes in high dimensional cases

PCP Tree: Handling high dimensional case

- Partition by PCA directions (not perpendicular to axes)
- Work much better on real data.

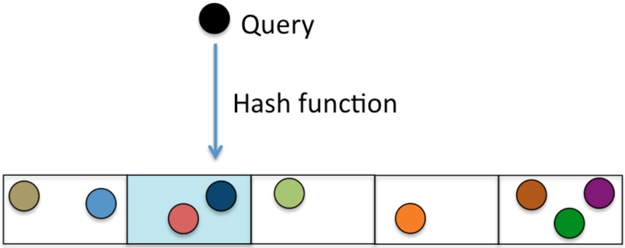


(Picture from
http://andrewd.ces.clemson.edu/courses/cpsc805/references/nearest_search.pdf)

Locality Sensitive Hashing

Locality Sensitive Hashing (LSH)

- Main Idea:
  - Randomly hash data to a small number of bins
  - Nearest points will have high probability to lie in the same bin
  - For a query point, search for points only within the same bin



Definition of LSH

- Let $\mathcal{F}$ be a family of functions that map elements from input space to bucket $\{1, 2, \ldots, s\}$.
- $\mathcal{F}$ is an LSH family: if a hash function h chosen uniformly at random from $\mathcal{F}$:
  - If $d(p, q) \leq R$, then $h(p) = h(q)$ with probability $\geq P_1$,
  - If $d(p, q) \geq cR$, then $h(p) = h(q)$ with probability $\leq P_2$.
- Such family $\mathcal{F}$ is called $O(R, cR, P_1, P_2)$-sensitive $(P_1 > P_2)$.

LSH-amplification

- Amplification:
  - AND operation of k functions:

$$\mathrm{AND}_{h_1,\ldots,h_k}(p) = \mathrm{AND}_{h_1,\ldots,h_k}(q) \;\text{ iff }\; h_i(p) = h_i(q),\;\forall i = 1,\ldots,k$$

$$(R, cR, P_1, P_2)\text{-LSH} \Rightarrow (R, cR, P_1^k, P_2^k)\text{-LSH}$$

  - OR operation of m functions:

$$\mathrm{OR}_{h_1,\ldots,h_k}(p) = \mathrm{OR}_{h_1,\ldots,h_k}(q) \;\text{ iff }\; h_i(p) = h_i(q) \text{ for some } i$$

$$(R, cR, P_1, P_2)\text{-LSH} \Rightarrow (R, cR, 1-(1-P_1)^m, 1-(1-P_2)^m)\text{-LSH}$$

  - Combined together:

$$(R, cR, P_1, P_2) - LSH \Rightarrow (R, cR, 1-(1-P_1^k)^m, 1-(1-P_2^k)^m) - LSH$$

- Example:

$$(0.8, 0.5) \Rightarrow (1-(1-0.8^8)^{15}, 1-(1-0.5^8)^{15}) = (0.93, 0.057)$$

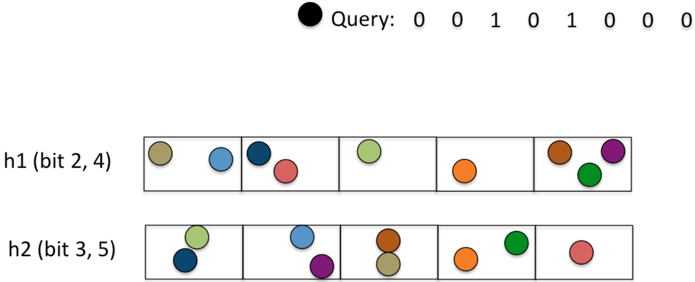
LSH: Hamming distance

- Each point $\boldsymbol{x} = \{0, 1\}^d$
- Hash function: sample a bit and map to 0/1

x1	0	1	0	0	1	0	0	1
x2	0	1	1	0	0	1	0	0
x3	0	0	1	0	1	0	0	0

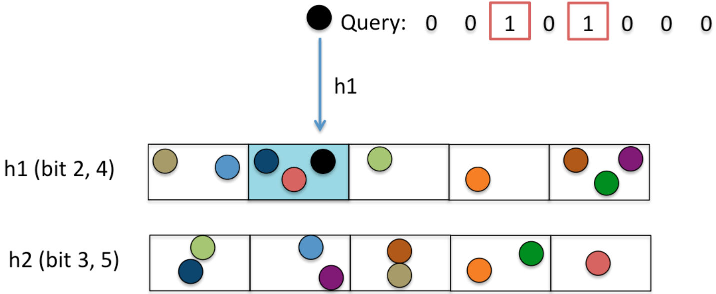
LSH: Hamming distance

- For each h_i, sample k elements and map to 2^k bins
- OR-function of $h_1, \ldots, h_m$
- Two points "collide" if $h_i(p) = h_i(q)$ for some $i = 1, \ldots, m$
- Linear search on all the points that collide with the query



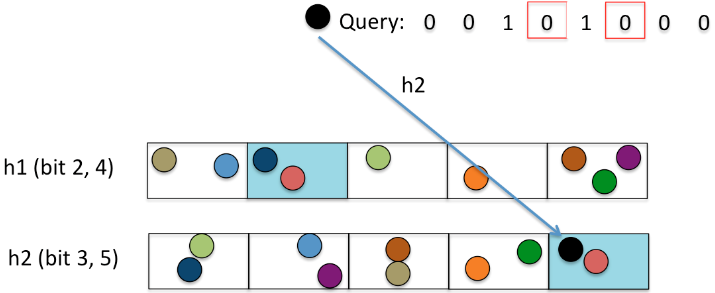
LSH: Hamming distance

- For each h_i, sample k elements and map to 2^k bins
- OR-function of $h_1, \ldots, h_m$
- Two points "collide" if $h_i(p) = h_i(q)$ for some $i = 1, \ldots, m$
- Linear search on all the points that collide with the query



LSH: Hamming distance

- For each h_i, sample k elements and map to 2^k bins
- OR-function of $h_1, \ldots, h_m$
- Two points "collide" if $h_i(p) = h_i(q)$ for some $i = 1, \ldots, m$
- Linear search on all the points that collide with the query



LSH: Hamming distance

- For each h_i, sample k elements and map to 2^k bins
- OR-function of $h_1, \ldots, h_m$
- Two points "collide" if $h_i(p) = h_i(q)$ for some $i = 1, \ldots, m$
- Linear search on all the points that collide with the query

LSH: Random Projections

- Hash function that preserves Euclidean distance:

$$h_{\boldsymbol{u},b}(\boldsymbol{x}) = \lfloor \frac{\boldsymbol{u}^T \boldsymbol{x} + b}{w} \rfloor$$

  where $\lfloor \cdot \rfloor$ is the floor operation, w is the width of a bucket, $\boldsymbol{u}$ sampled from $N(0, 1)$, b sampled from $(0, w)$
- Amplified by AND/OR operations.

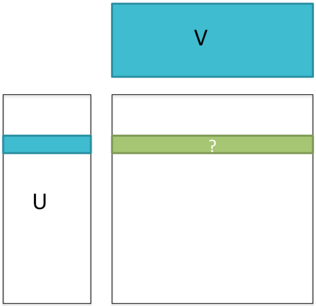
Summary

	Query Time	Space Used	Perprocessing Time
KD-Tree	$O(2^d \log n)$	$O(n)$	$O(nd \log n)$
LSH	$O(kmd + \text{linear search})$	$O(mkn)$	$O(mknd)$

Maximum Inner Product Search (MIPS)

MIPS: Motivation

- Matrix factorization models for Recommender systems:
  - $\boldsymbol{u}_i$: latent features for user $i = 1, \dots, m$
  - $\boldsymbol{v}_j$: latent features for item $j = 1, \dots, n$
  - $\boldsymbol{u}_i^T \boldsymbol{v}_j$: approximate the rating R_{ij}
- Prediction: Recommend a item to user 1
  - Select $j^* = \text{argmax}_{j=1,\dots,n} \boldsymbol{u}_1^T \boldsymbol{v}_j$
  - Linear search: $O(nk)$ time
- Similar application in multiclass classification.



Maximum Inner Product Search (MIPS)

- Given a database with n candidate vectors $\mathcal{X} := \{\boldsymbol{x}_i \in \mathbb{R}^d, i = 1, \ldots, n\}$.
- Given a query vector $\boldsymbol{v}$, find

$$\arg\max_{\boldsymbol{x}_i \in \mathcal{X}} \boldsymbol{v}^T \boldsymbol{x}_i$$

- Linear search: $O(nd)$ time
- Can we reduce the query time?

Maximum Inner Product Search (MIPS)

- Cannot directly use LSH for Euclidean distance
  - $\|\boldsymbol{v} - \boldsymbol{x}_i\|^2 = \|\boldsymbol{v}\|^2 + \|\boldsymbol{x}_i\|^2 - 2\boldsymbol{v}^T\boldsymbol{x}_i$
  - $\|\boldsymbol{v}\|^2$ is fixed for a query point, but $\|\boldsymbol{x}_i\|^2$ can vary
  - Nearest neighbor search: return

$$\arg\max_{\boldsymbol{x}_i} 2\boldsymbol{v}^T\boldsymbol{x}_i - \|\boldsymbol{x}_i\|^2$$

Maximum Inner Product Search (MIPS)

- Cannot directly use LSH for Euclidean distance
  - $\|\boldsymbol{v} - \boldsymbol{x}_i\|^2 = \|\boldsymbol{v}\|^2 + \|\boldsymbol{x}_i\|^2 - 2\boldsymbol{v}^T\boldsymbol{x}_i$
  - $\|\boldsymbol{v}\|^2$ is fixed for a query point, but $\|\boldsymbol{x}_i\|^2$ can vary
  - Nearest neighbor search: return

$$\arg\max_{\boldsymbol{x}_i} 2\boldsymbol{v}^T\boldsymbol{x}_i - \|\boldsymbol{x}_i\|^2$$

- Assymetric reduction to NN: define the mappings $P(\cdot), Q(\cdot)$ such that

$$\|P(\boldsymbol{x}_i) - Q(\boldsymbol{v})\|^2 \text{ proportional to } -\boldsymbol{x}_i^T\boldsymbol{v}$$

We can then find MIPS by

$$\operatorname*{argmin}_{\boldsymbol{x}_i} \|P(\boldsymbol{x}_i) - Q(\boldsymbol{v})\|^2$$

Reduction to NN: L2-ALSH

- Proposed in (Shrivastava and Li, "Asymmetric LSH (ALSH) for Sublinear Time Maximum Inner Product Search (MIPS)"). NIPS 2014
- Define

$$P(\boldsymbol{x}) = [\boldsymbol{x}; \|\boldsymbol{x}\|^2; \|\boldsymbol{x}\|^4; \cdots; \|\boldsymbol{x}\|^{2^m}] \quad Q(\boldsymbol{v}) = [\boldsymbol{v}; \frac{1}{2}; \frac{1}{2}; \cdots; \frac{1}{2}]$$

  So we have

$$\|P(\boldsymbol{x}_i) - Q(\boldsymbol{v})\|^2 = -2\boldsymbol{v}^T\boldsymbol{x}_i + \|\boldsymbol{v}\|^2 + \frac{m}{4} + \|\boldsymbol{x}_i\|^{2^{m+1}}$$

- By normalizing the database, we can make $\max_i \|\boldsymbol{x}_i\| < 1$

  $\|\boldsymbol{x}_i\|^{2^{m+1}} \to 0$ when $m \to \infty$
- Using this mapping, MIPS $\approx$ NN-search when m is large (applying LSH on $P(\boldsymbol{x}_1), \cdots, P(\boldsymbol{x}_n)$ and query $Q(\boldsymbol{v})$)

Reduction to NN: A better approach

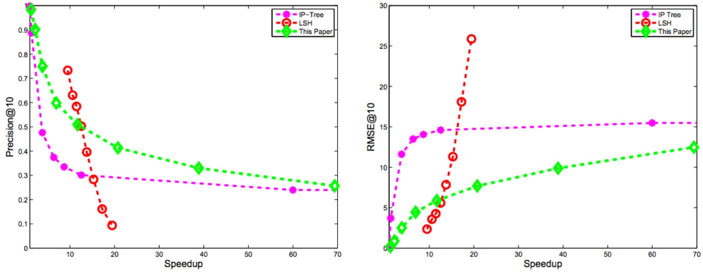
- SIMPLE-LSH (discussed in (Neyshabur and Srebro, "On Symmetric and Asymmetric LSHs for Inner Product Search". ICML 2015))
- Original proposed in (Bachrach et al., "Speeding Up the Xbox Recommender System Using a Euclidean Transformation for Inner-Product Spaces". Recsys, 2014.)
- A simple way to define $P(\cdot)$ and $Q(\cdot)$:

$$P(\boldsymbol{x}) = [\boldsymbol{x}; \sqrt{M - \|\boldsymbol{x}\|^2}] \quad Q(\boldsymbol{v}) = [\boldsymbol{v}; 0]$$

  where $M = \max_i \|\boldsymbol{x}_i\|^2$
- Easy to see $\|P(\boldsymbol{x}_i) - Q(\boldsymbol{v})\|^2 = M + \|\boldsymbol{v}\|^2 - 2\boldsymbol{x}_i^T\boldsymbol{v}$
- LSH with $P(\boldsymbol{x}_1), \cdots, P(\boldsymbol{x}_n)$ and query $Q(\boldsymbol{v})$

Some Comparisons

- Reduction to NN + PCA-tree (Bachrach et al., 2014).



(a) Precision vs. Speedup

(b) RMSE vs. Speedup

(Figure from Bachrach et al., 2014)

Directly Solving MIPS: Sampling Approach

- Assume $\boldsymbol{v}$ and $\{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n\}$ are all nonnegative.
- Sampling approach: sample according to the probability

$$P(i) \sim \boldsymbol{x}_i^T \boldsymbol{v} = \sum_t X_{it} v_t$$

- We can sample by the joint distribution

$$\begin{aligned} P(i,t) \sim v_t X_{it} &= v_t X_{it} / (\sum_i \sum_t X_{it} v_t) \\ &= \frac{v_t(\sum_i X_{it})}{\sum_t v_t(\sum_i X_{it})} \frac{X_{it}}{\sum_i X_{it}} \\ &= P_{\boldsymbol{v}}(t) P_t(i) \end{aligned}$$

And then $P(i) = \sum_t P(i,t)$.

Directly Solving MIPS: Sampling Approach

- Sampling MIPS: pre-processing
  - Compute $R_i = \sum_i X_{it}$
  - Construct distributions $P_t(i) \sim \frac{X_{it}}{\sum_i X_{it}}$ for all t.
- Query: (normalize $\boldsymbol{v}$)
  - Construct distribution $P_{\boldsymbol{v}}(t) \ \sim v_t R_t$
  - Sample many (i, t) pairs by (1) sampling t by $P_{\boldsymbol{v}}(t)$ (2) sampling j by $P_t(i)$
  - Output the top-T sampled index i.
- What's the time complexity?

Diamond Sampling

- Proposed in (Ballard et al., "Diamond Sampling for Approximate Maximum All-pairs Dot-product (MAD) Search". ICDM 2015.)
- Sampling approach fro MAD problem:

$$\max_{i,j} |\boldsymbol{x}_i^T \boldsymbol{x}_j|$$

- Can handle negative values in $\boldsymbol{x}_i, \boldsymbol{x}_j$.

Fast Sampling Algorithms

Sampling from a given distribution

- Sampling from a discrete probability distribution with n indices with distribution

$$p_1,\ p_2, \ldots,\ p_n$$

(Assume this is normalized, so $\sum_i p_i = 1$)

- What's the time complexity for generating one sample?

Linear Search

- Randomly generate a number $s \in [0, 1]$ uniformly

  Constant Time
- Construct a cummulated distribution

$$c_0 = 0, c_1 = c_0 + p_1, c_2 = c_1 + p_2, \ldots, c_n = c_{n-1} + p_n$$

- Find the index i such that $s \in [c_{i-1}, c_i]$.
- $O(n)$ time

Binary Search

- Preprocessing:

  Construct cummulated distribution $c_0, c_1, \ldots, c_n$

  $O(n)$

- Sampling:

  Randomly generate a number $s \in [0, 1]$ uniformly

  Binary search to find i such that $s \in [c_{i-1}, c_i]$

  $O(\log n)$ time

Alias Method

- Preprocessing:

  Create an Alias Table

  $O(n)$ time (Vose's algorithm)

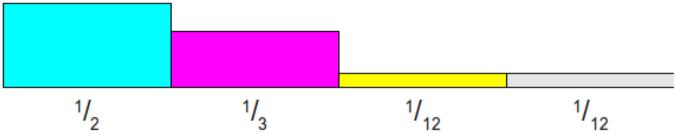
- Sampling:

  Generate a uniform random number in $\{1, 2, \ldots, n\}$

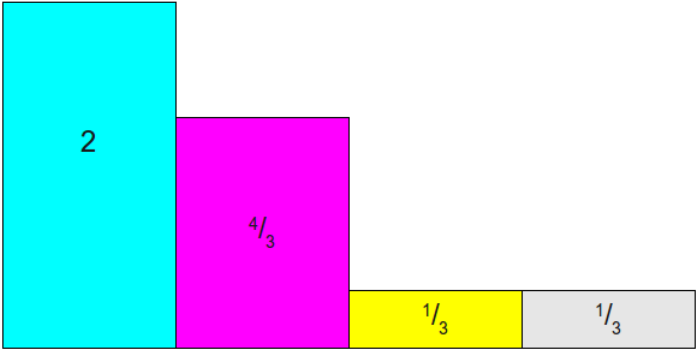
  Generate another random number to determine the index

  Constant time!

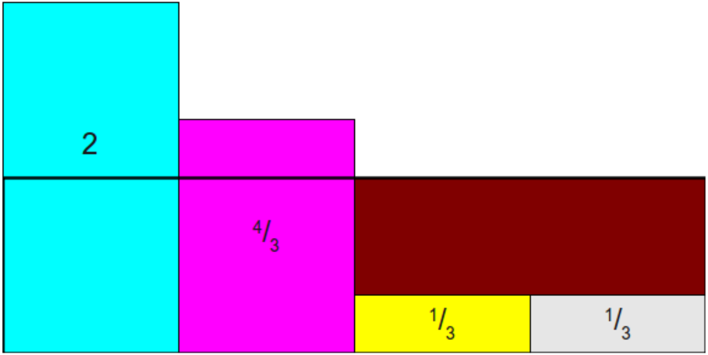
Alias Method



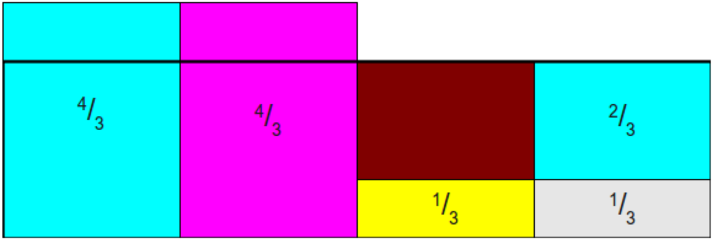
Alias Method



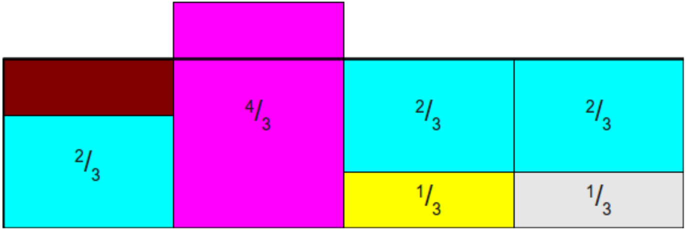
Alias Method



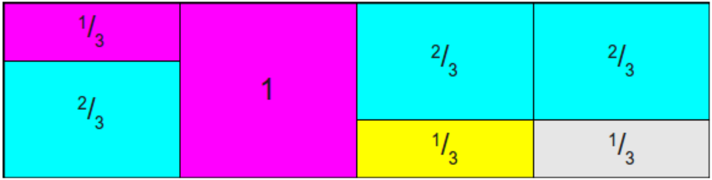
Alias Method



Alias Method



Alias Method

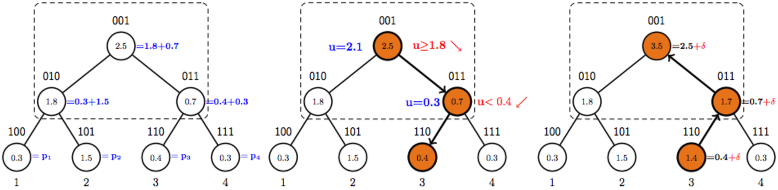


F+ Tree

- Sampling from distribution $p_1, \ldots, p_n$
- But one of p_i will change at each time (or once every few iterations)
- Alias Table does not work (need to reconstruct the whole table)
- F+ Tree (Fenwick Tree)
  - Sampling: $O(\log n)$ time
  - Update: $O(\log n)$ time

F+ Tree

	Data Structure Space	Initialization Time	Initialization Space	Generation Time	Parameter Update Time
LSearch	$c_T = \mathbf{p}^\top \mathbf{1}$: $O(1)$	$O(T)$	$O(1)$	$O(T)$	$O(1)$
BSearch	$\mathbf{c} = \mathtt{cumsum}(\mathbf{p})$: $O(T)$	$O(T)$	$O(1)$	$O(\log T)$	$O(T)$
Alias Method	*prob*, *alias*: $O(T)$	$O(T)$	$O(T)$	$O(1)$	$O(T)$
F+tree Sampling	$\mathtt{F.initialize}(\mathbf{p})$: $O(T)$	$O(T)$	$O(1)$	$O(\log T)$	$O(\log T)$



(a) F+tree for $\mathbf{p} = [0.3, 1.5, 0.4, 0.3]^\top$ (b) Sampling (c) Updating (with $\delta = 1.0$)

Coming up

Questions?