L13 -- A-Priori Example + Bloom Filters + Quantiles
[Jeff Phillips - Utah - Data Mining]

---------------------------------
A-Priori Example

T1 = {1,2,6,7}
T2 = {1,2,5,6}
T3 = {2,3,4,6}
T4 = {1,6,7}
T5 = {2,6}

{n choose k} sets of size k  ~  n^k  (too large)

Threshold 40%  ->  2/5

1: 3
2: 3
3: 2
4: 1 <discard>
5: 1 <discard>
6: 4
7: 2

1+2: 2
1+3: 0 <discard>
1+6: 3
1+7: 2
2+3: 1 <discard>
2+6: 3
2+7: 1 <discard>
3+6: 1 <discard>
3+7: 0 <discard>
6+7: 2

1+2+6: 2
1+6+7: 2

none at size 4 possible.

Maximal (at 40%): (1+2+6) (1+6+7) (3)

-----------------------------------------------
-----------------------------------------------

Streaming Algorithms

Stream : A = <a1,a2,...,am>
  ai in [n]  size log n
Compute f(A) in poly(log m, log n) space
   "one pass"

Let f_j = |{a_i in A | a_i = j}|
F_1 = sum_j f_j  = m == total count

---------------------------------
Bloom Filters

Maintain set S subset [u]
 allow false positives
    no fasle negatives

Initialize Array B of n bits all 0
have k hash functions {h1, h2, ..., hk} in \H

Put a_i in A in set S:
 for j = 1 to k
   set B[hj(a_i)] := 1

Check if a_i in A in set S:
  for j = 1 to k
   if (B[hj(a_i)] == 0) --> return NO
  return YES

*** No false negatives
*** Some false positives

------------
Analysis:
m bits:
n items


probability a bit not set to 1 by 1 hash function:
  1-1/m

probability bit not set to 1 by k hash functions:
 (1-1/m)^k

on inserting n elements, probability a bit is 0:
 (1-1/m)^{kn}      (*)

on inserting n elements, probability a bit is 1:
$1 - (1-1/m)^{kn}$

probability of false positive:
$(1 - (1-1/m)^{kn})^k$
$\approx$
$(1 - e^{-kn/m})^k$

[(*) not quite right, assumes independence of bits being set ]

So what is the "right" value of k?
$k \approx (m/n) \ln 2$

-----------------------------------------------
-----------------------------------------------
Quantiles:

Let [u] be an ordered set.
Let A be multiset in [u] size n

Quantile:
Given x in [u]
-> $A_x = \{a \text{ in } A \mid a \le x\}$
-> $|A_x|/n$

eps-quantile:
for any x in [u]
--> return q(x) s.t.
$| q(x) - |A_x|/n | \le eps$

*** Like a histogram ***

----------
Old best algorithm: Greenwald-Khanna

Maintain set of break points:
{b1, b2, ..., b_k} such that know approximate
q(bj) for each bj
sometimes insert new points,
occasionally delete old points if too dense

works ok, but very complicated analysis.
$k = O((1/eps) \log(eps\ n) \log(u))$
---------
New best algorithm: mergeable summaries

Maintain set S of $k \sim (1/eps)$ points

```
 q(x) = |S_x| / k
 each point "worth" 1/k

merge two summaries S1, S2
  -->  sort S1 cup S2
    size = 2k
  --> reduce size, pick all even points or all odd points
      + unbiased
      + size k
  If k = O((1/eps) sqrt(log(1/eps))) error does not grow

------
But what if |S1| != |S2|?
  let N = 2^s for smallest s s.t. 2^s >= n
store:
 S as h = log(1/eps) levels
  level l in [log(1/eps)]
  level represents N/(2^l) points
  level h+1 is random "buffer" : sample of constant size
 Each level is size k = O((1/eps) sqrt(log(1/eps)))
  levels are either empty or full

 on merge, merge equal weight levels.
  + points only "move down" in levels

Size now k*h = O((1/eps) log^{3/2} (1/eps))

Streaming:  each new point is merged into random buffer
```