L16 -- Random Projection
[Jeff Phillips - Utah - Data Mining]

Two techniques:
- random projections to subspace (data independent)
- basis selection

P in R^d   and |P| = n
goal: mu : P -> R^k   (k << d)
s.t. max_{p,q in P}
(1-eps) ||p-q|| <= ||mu(p) - mu(q)|| <= (1+eps) ||p-q||

Idea: randomly project the data to a subspace.

How to get a random vector?   ???
1. compute random Gaussian variable x_i in R^d
2. normalize to u_i = x_i/||x_i||

Then ~mu(y_i) = <p, u_i>

Lets focus on simpler problem for now:
for one p in P (s.t. ||p|| = 1)
(1-eps/2) ||p||^2 <= ||mu(p)||^2 <= (1+eps/2)||p||^2

sqrt{(1-eps/2)} > (1-eps)   and sqrt{(1+eps/2)} < (1-eps)
pretend just eps/2 = eps ...

||p||^2 = sum_{i=1}^d ||p_i||^2

But, it has the same problem as homework.
E[||~mu(p)||^2] == ???
||p||^2/d <--- too small

let mu(p) = ~mu(p) * d
now E[||mu(p)||^2] = ||p||^2

Worst case ||mu(p)||^2 - ||p||^2 <= (d-1) ||p||^2 = Delta_i
Var[||mu(p)||^2] = 1

Can use Chernoff Bound
- expected value = 0
- bounded variance [or bounded worst case]

Choose k random directions {u_1, u_2, ..., u_k} <-- basis
mu(p)_i = <p, u_i> * sqrt{d/k}

mu(p) in R^k
||mu(p)||^2 = sum_{i=1}^k ||mu(p)_i||^2

E[||mu(p)||^2 - ||p||^2] = 0
E[||mu(p)_i||^2 - ||p||^2/k] = 0
Var[||mu(p)||^2] <= ||p||
Var[||mu(p)_i||^2] = ||p||/k
Var_i = Var[||mu_i(p)||^2/||p||^2] = 1/k

Pr[| ||mu(p)||^2 - ||p||^2 | > eps ||p||^2] =
Pr[| ||mu(p)||^2/||p||^2 - 1 | > eps] <
      2 exp(- eps^2 / 4 sum_{i=1}^k Var_i^2) =
      2 exp(- eps^2 / 4 k (1/k^2))
      < delta'

   k eps^2 /4 = ln(2/delta')
   k = (4/eps^2) ln(2/delta'))

OK, so with k = c/eps^2 log(1/delta'), one norm is preserved.

now think of each ||p - q|| for p,q in P a norm that needs preserving
   with ||mu(p) - mu(q)|| = ||mu(p-q)||
   since mu is linear, then mu(p) - mu(q) = mu(p-q)

   {n choose 2} < n^2 such norms

   set delta' = delta/n^2

then chance that each norm has error is at most delta/n^2
  then chance any has norm error is sum_{i=1}^n^2 delta/n^2 = delta
    <<<<<< Union Bound >>>>>>>

So k = c/eps^2 log(n^2/delta)
     = O((1/eps^2) log (n/delta))

Problems:
 - not as good as SVD (optimal in some sense)
 - does not preserve dimension-structure
 - ignores data distribution
Advantages:
 + very easy to implement
 + ignores data distribution (oblivious)

```
 + can be implemented very fast (only need random {-1,0,+1} matrix)
 + if sparse -> no longer sparse (strangely, this prevents from being faster)


--------------------------------------------------------
Column sampling

 - returns set or t = (1/eps^2) k log k dimensions that is close to best k
from SVD.

---------
simple
  compute w(j) = ||p_j||^2 of each column.
  Select column proportional to w(j)
       <<<<<<< just like k-means++ >>>>>>>>
  assume that columns picked are j on J and |J| = t

  set mu(p)_i = p_j * 1/w(j) * (d/t)
  -->  mu(P) = Q_t

P = U S V^T = [U_k U_k^#] [S_k 0; 0 S_k^#] [V_k ; V_k^#]
            = U_k S_k V_k^T + U_k^# S_k^# (V_k^#)^T
P_k = U_k S_k V_k^T

  -> gives weak approximation, but very easy.
  -> can do both rows and columns to get both subspace and "coreset"

  ||P - mu(P)||_2^2 = sum_{p in P} ||p - mu(p)||_2^2
  ||P - mu_k(P)||_2^2 = sum_{p in P} ||p - mu_k(p)||_2^2
    where mu_k is the best linear rank-k projection (from SVD)

  ||P - Q_t||_2^2 <= ||P - P_k||_2^2 + eps ||P||_F^2
and
  ||P - Q_t||_F^2 <= ||P - P_k||_F^2 + eps ||P||_F^2

  Frobenious norm:  ||P||_F^2 = sum_{i=1}^n ||p_i||_2^2



--------------------
Better result:
  1.  Construct V_k^T   <--- subspace of the best rank-k approximation
                             defines mu_k( )
  2.  Let w'(j) = ||(V_k^T)_j||^2 = sum_{p in P} (<mu_k(p), x_i>)^2
  3.  Select t = (1/eps^2) k log k columns: J
      mu'(p)_i = p_j * 1/w'(j) * (d/t)
      mu'(P) = Q'_t
```

```
Now:
  ||P - Q_t||_F^2 <= ||P - P_k||_F^2 + eps ||P - P_k||_F^2
  ||P - Q_t||_F^2 <= (1+eps)||P - P_k||_F^2

   -> gives better approximation
   -> takes about as long as SVD_k, but gives better result

---------------------
t = (1/eps^2) k log k
    (1/eps^2) comes from Chernoff bound, need to bound error
    k log k   comes from Coupon Collector, need to hit each top k component
```