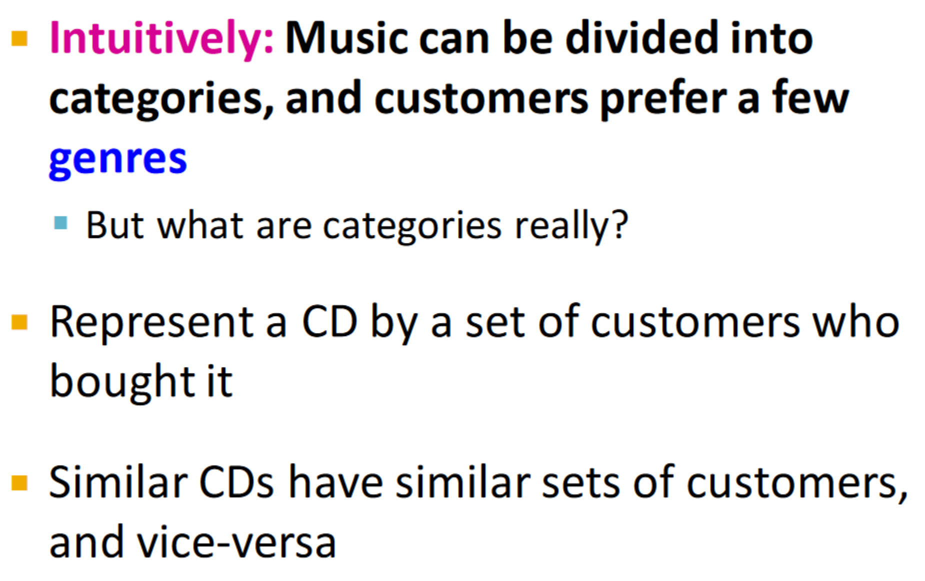
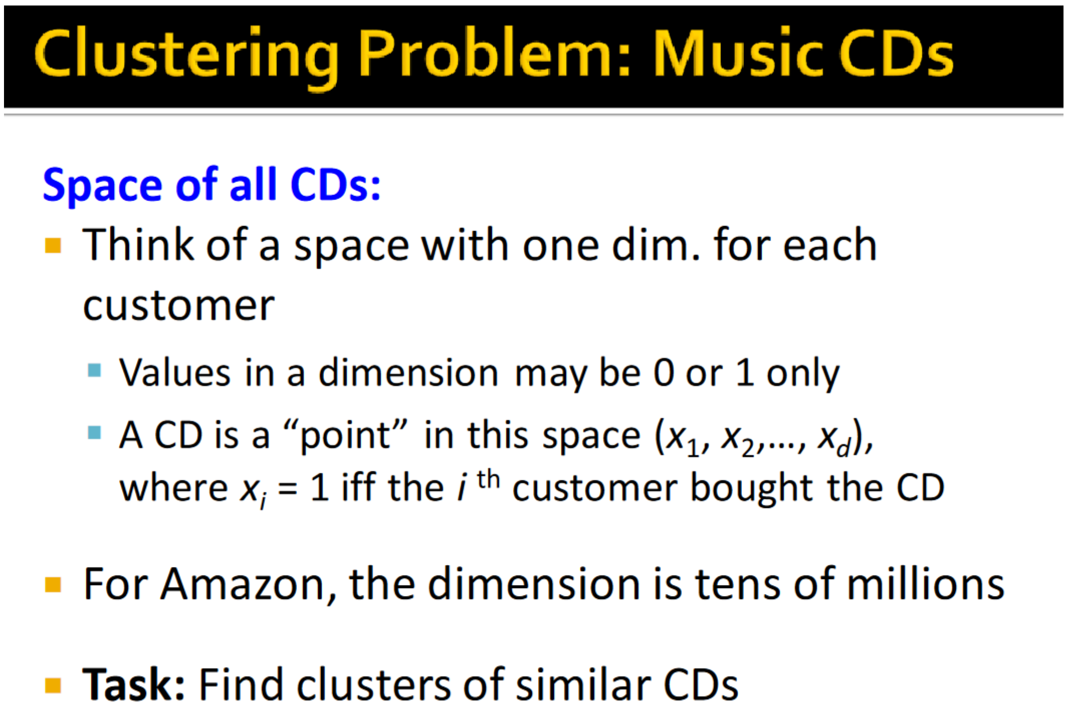
part3 Clustering and Dimension Reduction

CS246对于聚类和降维的介绍更多注重计算方法和在大数据场景下的应用。聚类主要介绍K-Means和BFR，降维介绍SVD及其计算方法和应用。



- **Intuitively: Music can be divided into categories, and customers prefer a few genres**
  - But what are categories really?
- Represent a CD by a set of customers who bought it
- Similar CDs have similar sets of customers, and vice-versa



Clustering Problem: Music CDs

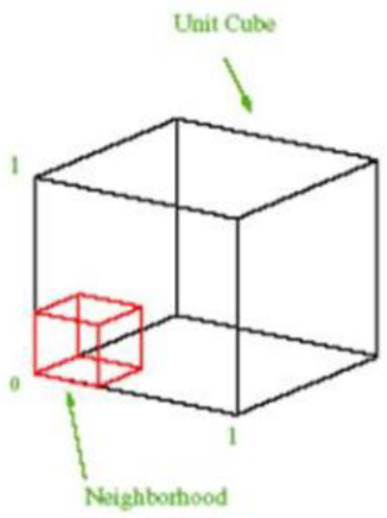
Space of all CDs:

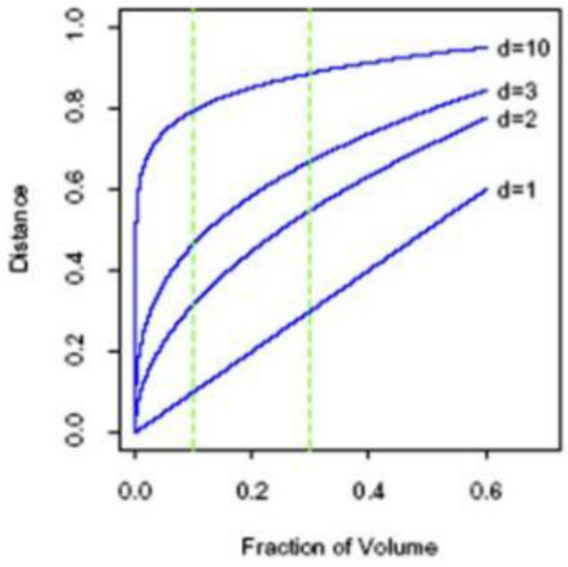
- Think of a space with one dim. for each customer
  - Values in a dimension may be 0 or 1 only
  - A CD is a "point" in this space $(x_1, x_2, \ldots, x_d)$, where $x_i = 1$ iff the i th customer bought the CD
- For Amazon, the dimension is tens of millions
- **Task:** Find clusters of similar CDs

给出用户购买CD的场景，我们可以认为用户天生的音乐喜好有不同的Type，CD也对应着不同类型的音乐种类。如果用户的喜好和CD的种类匹配，用户的购买可能就会更高。我们可以将CD space的点定义为i个维度的$customer\ i$是否购买，即01向量。我们的目标是对于这些点进行聚类找到相同类型的CD。

但是对于高维数据的聚类往往是困难的： 维度诅咒。

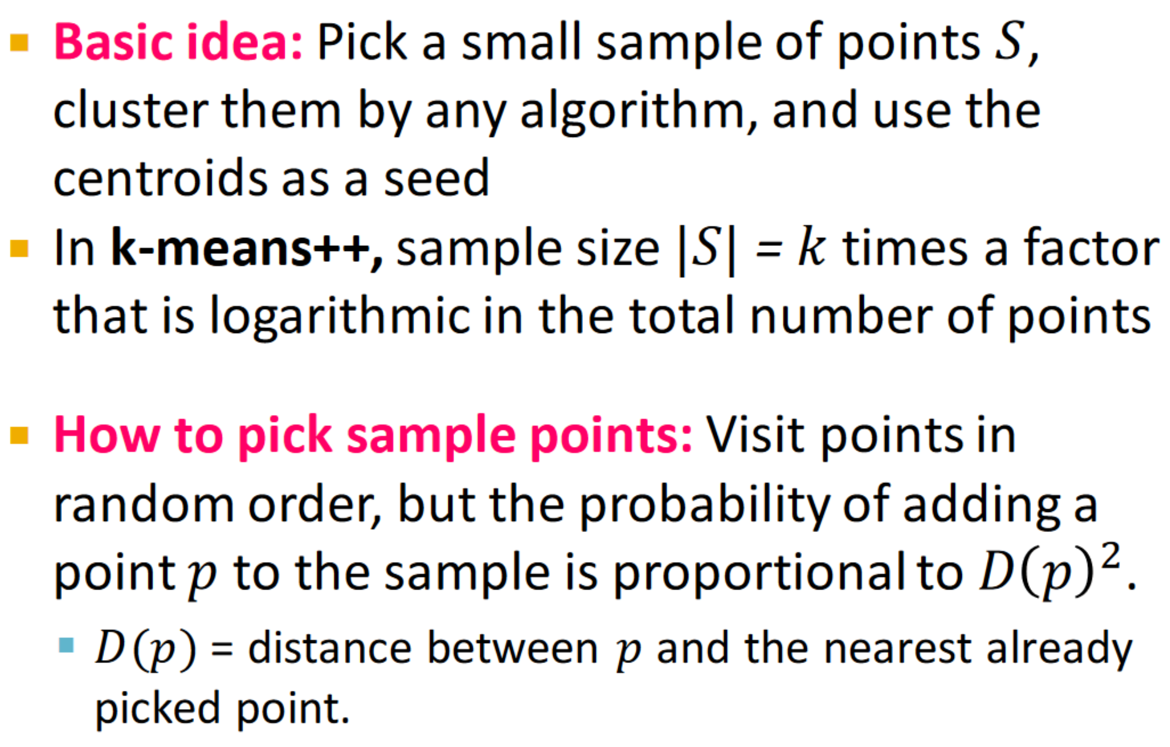
Curse of Dimensionality: All points are very far from each other





假设我们有10000个点均匀分布在（0， 1）上，那么在一维空间中，如果我们想要覆盖0.1%的数据，即10个点。我们需要10/10000 = 0.001的距离。在二维空间中，我们需要$\sqrt{0.001} = 0.032$的距离。在d维空间中，变为$(0.001)^{1/d}$。那么随着维度的升高，我们需要覆盖同样比例数据时所需要的距离也在不断升高。换句话说，在高维空间内，所有的点都倾向于远离其他点，即距离丧失了意义。

K-Means and BRF

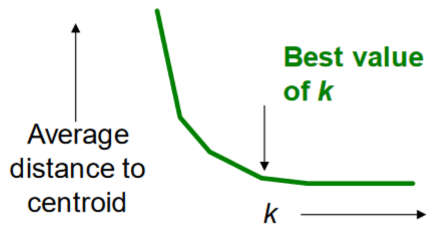


- **Basic idea:** Pick a small sample of points S, cluster them by any algorithm, and use the centroids as a seed
- In **k-means++,** sample size $|S|$ = k times a factor that is logarithmic in the total number of points
- **How to pick sample points:** Visit points in random order, but the probability of adding a point p to the sample is proportional to $D(p)^2$.
  - $D(p)$ = distance between p and the nearest already picked point.

K-Means的具体步骤不再赘述。上图是K-Menas++的一些改进思路。K-Means++在选择样本点是不再使用随机抽样的方式。在选择一个centroids后，计算所有样本点和当前centroid的距离，以距离的平方为权重，即$D(p)$，进行加权随机抽样。

How to select *k*?

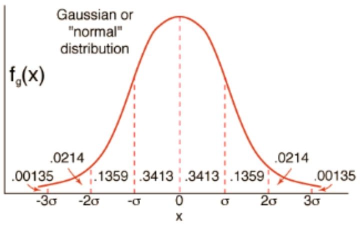
- Try different **k**, looking at the change in the average distance to centroid as **k** increases
- Average falls rapidly until right **k**, then changes little



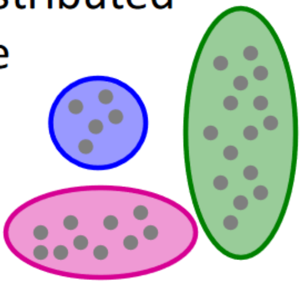
通过观察所有点到centroid的距离之和来选择聚类数目K。

K-Means有简单快速的优点，但是缺点也很明显。在大样本的情况下算法迭代需要遍历全部数据集，因此K-Means需要将全部的数据集读入内存中。这对于硬件要求较高，那么我们是否可以以损失部分精度为代价，部分读取数据来进行聚类计算呢?

BFR Algorithm

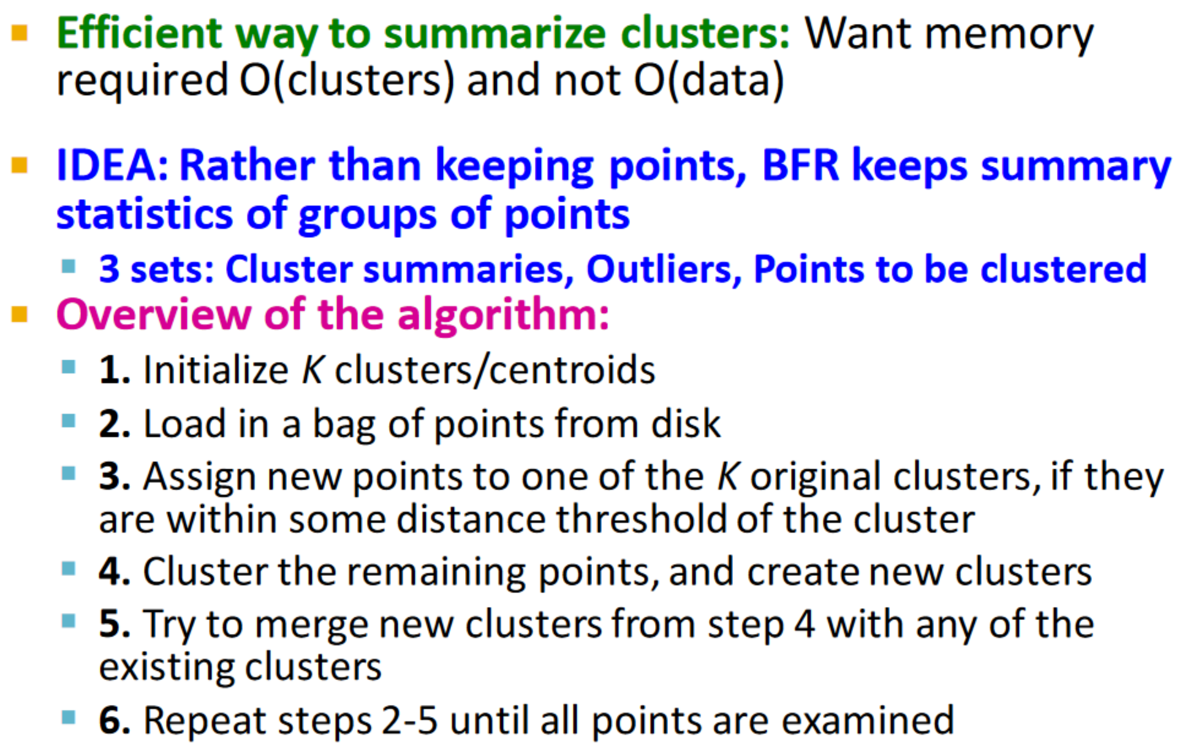


- **BFR** [Bradley-Fayyad-Reina] is a variant of *k*-means designed to handle **very large** (disk-resident) data sets
- **Assumes** that clusters are normally distributed around a centroid in a Euclidean space
  - Standard deviations in different dimensions may vary
    - Clusters are axis-aligned ellipses



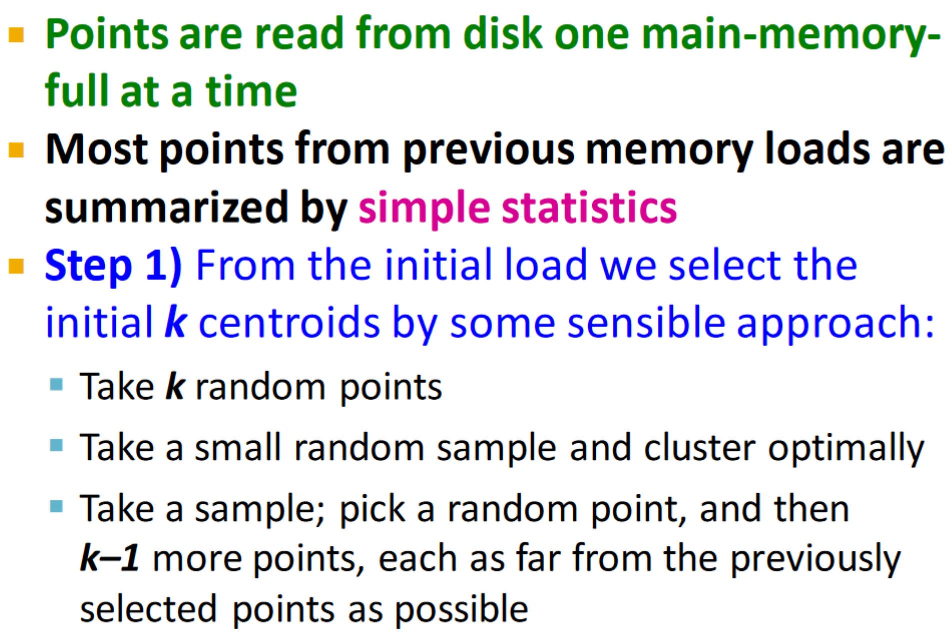
- Goal is to find cluster centroids; point assignment can be done in a second pass through the data.

BFR算法是K-Means的变种，用来应对海量数据的聚类任务。首先BFR对于数据的假设较强，他假设Clusters是围绕着某个centroid正态分布。



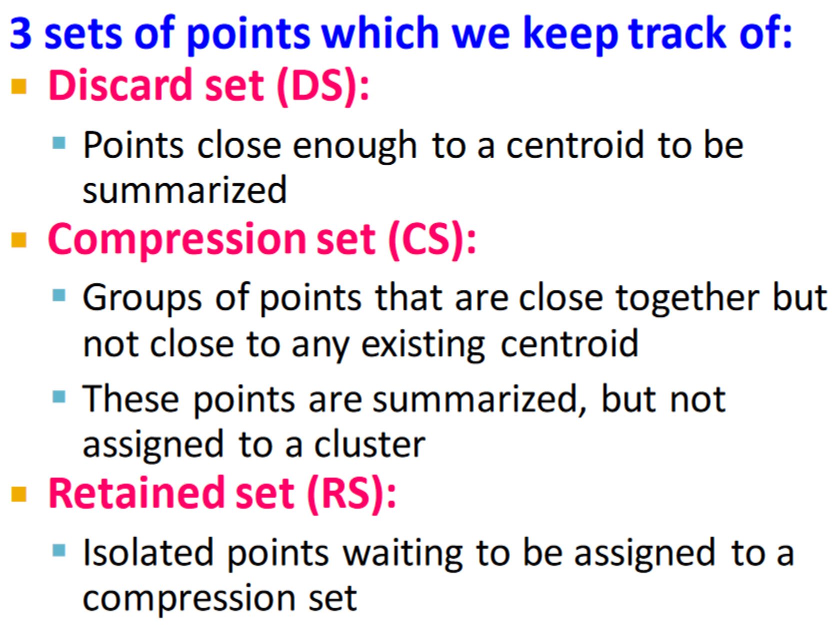
- **Efficient way to summarize clusters:** Want memory required O(clusters) and not O(data)
- **IDEA: Rather than keeping points, BFR keeps summary statistics of groups of points**
  - **3 sets: Cluster summaries, Outliers, Points to be clustered**
- **Overview of the algorithm:**
  - **1.** Initialize *K* clusters/centroids
  - **2.** Load in a bag of points from disk
  - **3.** Assign new points to one of the *K* original clusters, if they are within some distance threshold of the cluster
  - **4.** Cluster the remaining points, and create new clusters
  - **5.** Try to merge new clusters from step 4 with any of the existing clusters
  - **6.** Repeat steps 2-5 until all points are examined

BFR的内存需求为$O(clusters)$而非K-Means中的$O(data)$。为了节约内存空间，BFR将数据分为三个Sets并且每个set只保留他们相应的描述统计量。计算时，分批加载数据。



- **Points are read from disk one main-memory-full at a time**
- **Most points from previous memory loads are summarized by simple statistics**
- **Step 1)** From the initial load we select the initial ***k*** centroids by some sensible approach:
  - Take ***k*** random points
  - Take a small random sample and cluster optimally
  - Take a sample; pick a random point, and then ***k*–*1*** more points, each as far from the previously selected points as possible

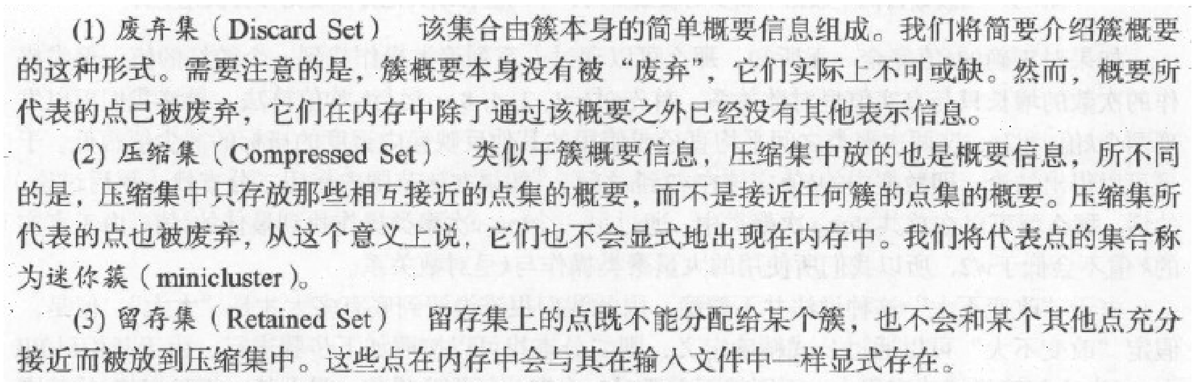
我们首先选取K个点作为centorids。上图给出了三种可能的方案。



3 sets of points which we keep track of:

- **Discard set (DS):**
  - Points close enough to a centroid to be summarized
- **Compression set (CS):**
  - Groups of points that are close together but not close to any existing centroid
  - These points are summarized, but not assigned to a cluster
- **Retained set (RS):**
  - Isolated points waiting to be assigned to a compression set

由于BFR不在内存中保留全部数据，因此我们将被踢出内存的数据归为以上三类：丢弃集，压缩集和留存集。



(1) 废弃集（Discard Set） 该集合由簇本身的简单概要信息组成。我们将简要介绍簇概要的这种形式。需要注意的是，簇概要本身没有被“废弃”，它们实际上不可或缺。然而，概要所代表的点已被废弃，它们在内存中除了通过该概要之外已经没有其他表示信息。

(2) 压缩集（Compressed Set） 类似于簇概要信息，压缩集中放的也是概要信息，所不同的是，压缩集中只存放那些相互接近的点集的概要，而不是接近任何簇的点集的概要。压缩集所代表的点也被废弃，从这个意义上说，它们也不会显式地出现在内存中。我们将代表点的集合称为迷你簇（minicluster）。

(3) 留存集（Retained Set） 留存集上的点既不能分配给某个簇，也不会和某个其他点充分接近而被放到压缩集中。这些点在内存中会与其在输入文件中一样显式存在。

既然废弃集或压缩集中的点不再存储，那么这个废弃集是如何表示这些点的信息的呢?
N，SUM，SUMSQ表示：N表示的点的数目,向量SUM表示 所有点在每一维的分量之和，即SUM[i]表示第i维上的分量和。向量SUMSQ表示所有点在每一维的分量的平方和。

因此，如果数据为d维的话，通过此方法可以用2d+1个值来表示一个废弃集或压缩集。我们的实际目标是将一系列点表示为它们的数目、质心和每一维的标准差。根据N-SUM-SUMSQ表示，我们可以得到：

$$\text{第}i\text{维的质心} = \frac{SUM_i}{N}$$

$$\text{第}i\text{维的方差} = \frac{SUMSQ_i}{N} - \left(\frac{SUM_i}{N}\right)^2$$

通过这种表示方式，如果当前Cluster需要新加入一个点时。N只需要增加1，同时我们可以直接将这个点的坐标加在SUM上求出新的SUM(element-wise)。将向量第i个分量的平方加载SUMSQ上计算出新的SUMSQ。而如果采取质心的方法，我们则需要先重新计算质心，再计算方差等。同时如果我们想合并两个集合，那么我们直接将这两个集合对应的N-SUM-SUMSQ相加即可。

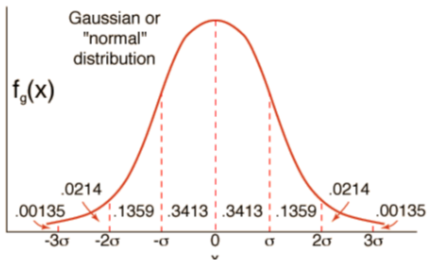
数据处理过程:

1. 首先找到所有**“sufficiently close”**充分接近某个簇质心的点加入到该簇中。加入新的点后，DS废弃集的调整计算通过N-SUM-SUMSQ表示很容易实现，直接加上这些新加入的点的N,SUM,$SUMSQ$值即可。
2. 剩下得即是那些并不充分接近任一个簇质心的点。我们将这些点和之前的留存集中的点一起进行聚类（use any main-memory clustering algorithm）。将聚类的点概括表示并加入到压缩集CS中，剩余的单点簇则表示为点的留存集RS。
3. 通过第二步后，现在我们有了新的迷你簇，它们和上一个块的数据处理后留下的压缩集迷你簇和留存集中间可能距离很近是可以合并的。因此这一步就是将这些迷你簇和以前的压缩集的迷你簇和以

前的留存集进行合并。
（如果这是最后一块数据了，那么就将这些留存集的点和压缩集分配到距离最近的簇中，否则继续保留压缩集和留存集，等待和下一块的数据一起处理。）
4. . 分配给一个簇或迷你簇的点，即不再留存集中的点会和分配结果一起写出到二级存储器中。

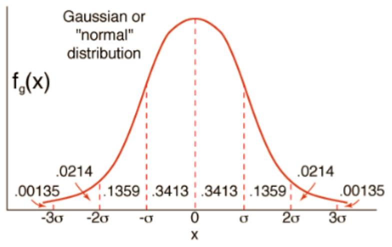
根据上述的流程，我们有两个定义需要明确。第一是如何定义sufficiently close。第二是如何定义压缩集的两个cluster是否应该合并。

- **Q1) We need a way to decide whether to put a new point into a cluster (and discard)**
- **BFR suggests two ways:**
  - The **Mahalanobis distance** is less than a threshold
  - **High likelihood of the point belonging to currently nearest centroid**



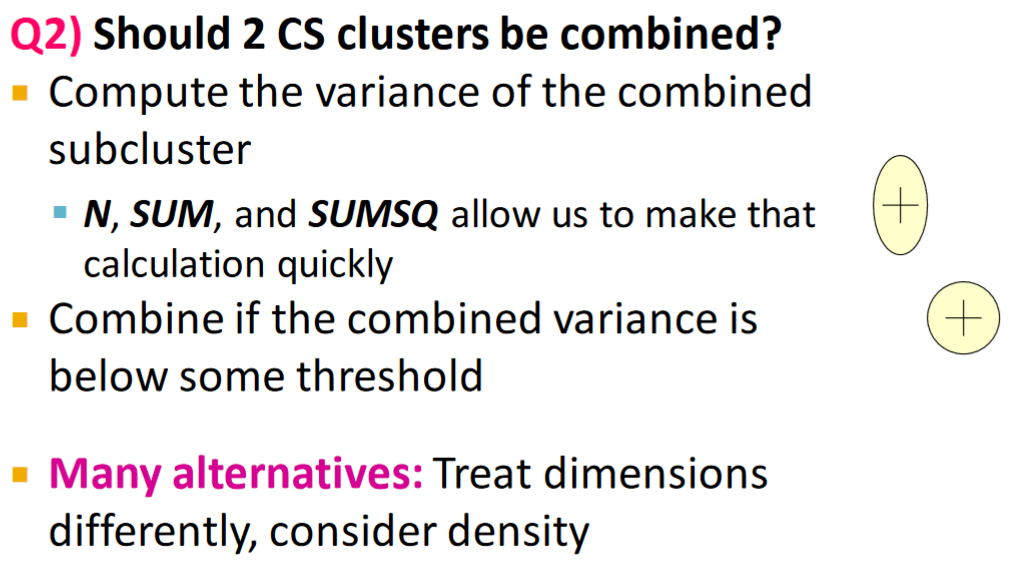
对于第一个问题BFR给出了两种解决方法。计算点到cluster质心的马氏距离，或者是根据点属于当前簇的概率。

- If clusters are normally distributed in ***d*** dimensions, then after transformation, one standard deviation = $\sqrt{d}$
  - i.e., 68% of the points of the cluster will have a Mahalanobis distance $< \sqrt{d}$
- Accept a point for a cluster if its M.D. is **<** some threshold, e.g. **2** standard deviations



马氏距离可以看做经过归一化后的欧氏距离，如果cluster在d个维度上是正态分布的，那么在归一化之后，数据的标准差为$\sqrt{d}$。我们可以根据3σ的原则定义阈值，如两倍的标准差为threshold。

关于第二个问题，BFR给出的解决方案是如果两个cluster合并后的方差小于阈值，即可以合并。



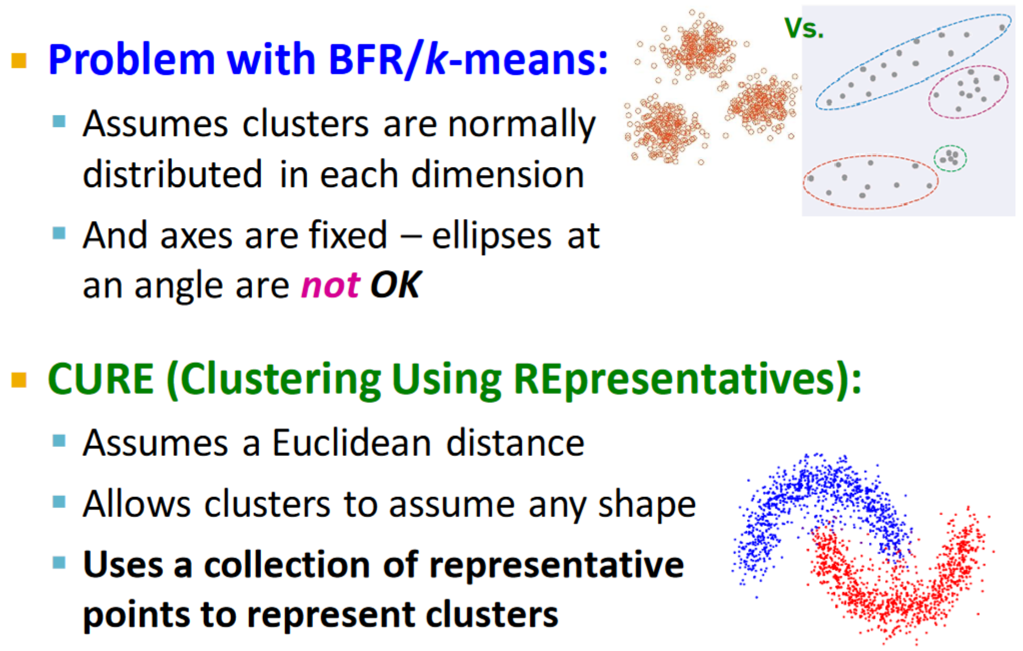
Q2) Should 2 CS clusters be combined?

- Compute the variance of the combined subcluster
  - ***N***, ***SUM***, and ***SUMSQ*** allow us to make that calculation quickly
- Combine if the combined variance is below some threshold

- **Many alternatives:** Treat dimensions differently, consider density

CURE

大规模聚类算法CURE算法（Clustering Using Representatives），仍然假定运行在欧式空间下。我们知道BFR算法对簇的分布和形状都有很强的假设条件，因此在实际使用中有许多并不满足条件的聚类情况。CURE算法不需要簇满足正态分布，更不需要符合轴平行，即对簇的形状没有任何假设。CURE算法使用一些代表点的集合来表示簇而不再是质心。



- **Problem with BFR/*k*-means:**
  - Assumes clusters are normally distributed in each dimension
  - And axes are fixed – ellipses at an angle are ***not OK***

- **CURE (Clustering Using REpresentatives):**
  - Assumes a Euclidean distance
  - Allows clusters to assume any shape
  - **Uses a collection of representative points to represent clusters**

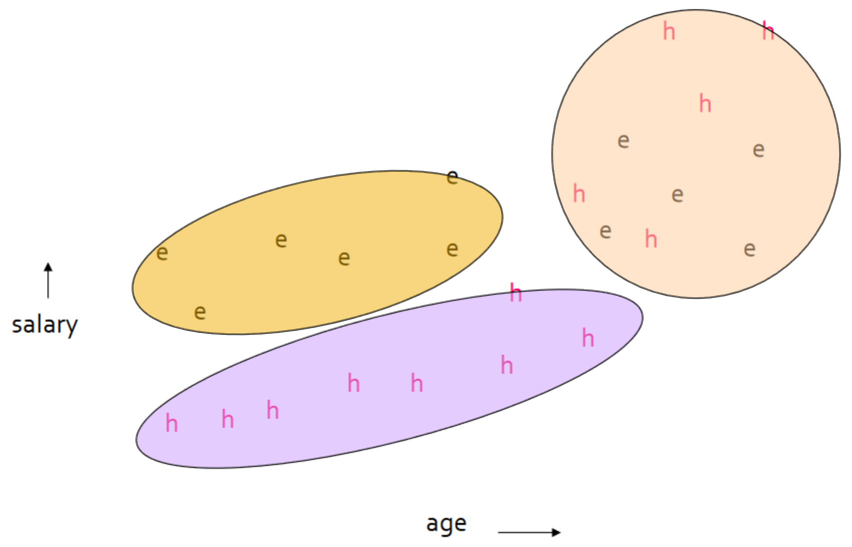
CURE分为2-passes进行计算。

2 Pass algorithm. Pass 1:

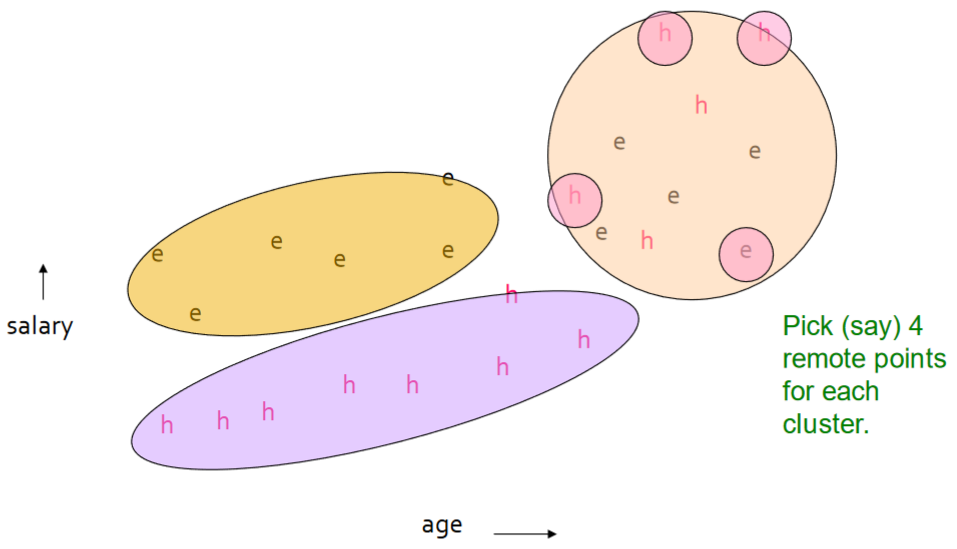
- **0) Pick a random sample of points that fit in main memory**
- **1) Initial clusters:**
  - Cluster these points hierarchically – group nearest points/clusters
- **2) Pick representative points:**
  - For each cluster, pick a sample of points, as dispersed as possible
  - From the sample, pick representatives by moving them (say) 20% toward the centroid of the cluster

聚类过程如下：

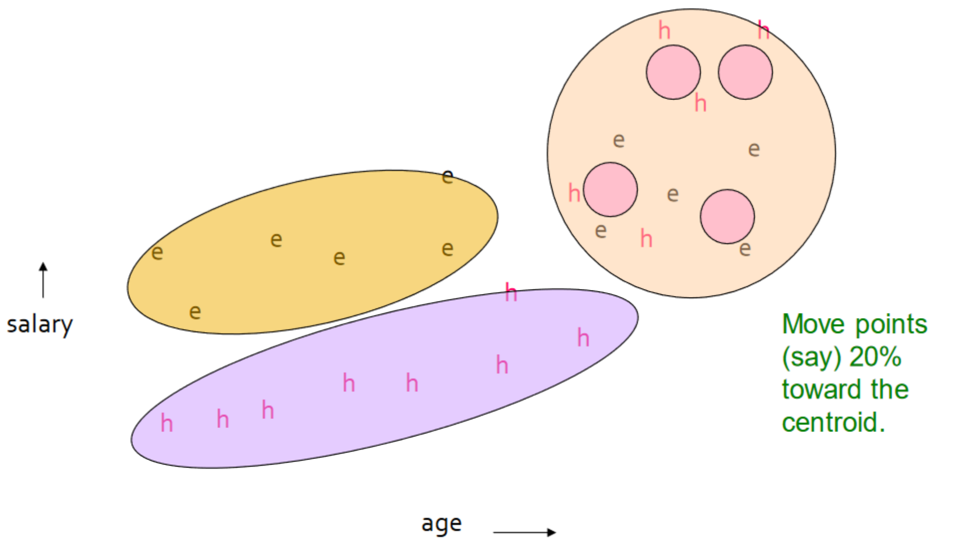
Example: Initial Clusters



Example: Pick Dispersed Points

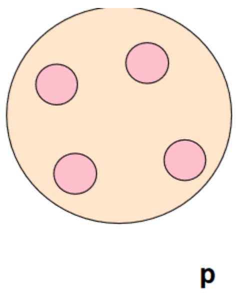


Example: Pick Dispersed Points



Pass 2:

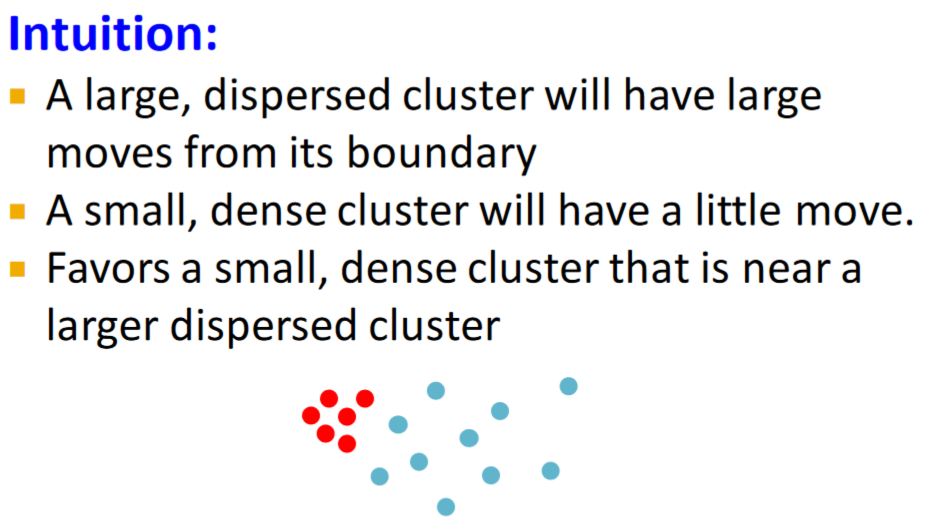
- Now, rescan the whole dataset and visit each point ***p*** in the data set
- **Place it in the "closest cluster"**
  - Normal definition of "closest": Find the closest representative point to ***p*** and assign it to representative's cluster



算法步骤

1. 抽取一部分样本数据在内存中进行聚类;
2. 从每个簇中选择一小部分点集作为簇的代表点（可以用 K-means 算法来选择点;

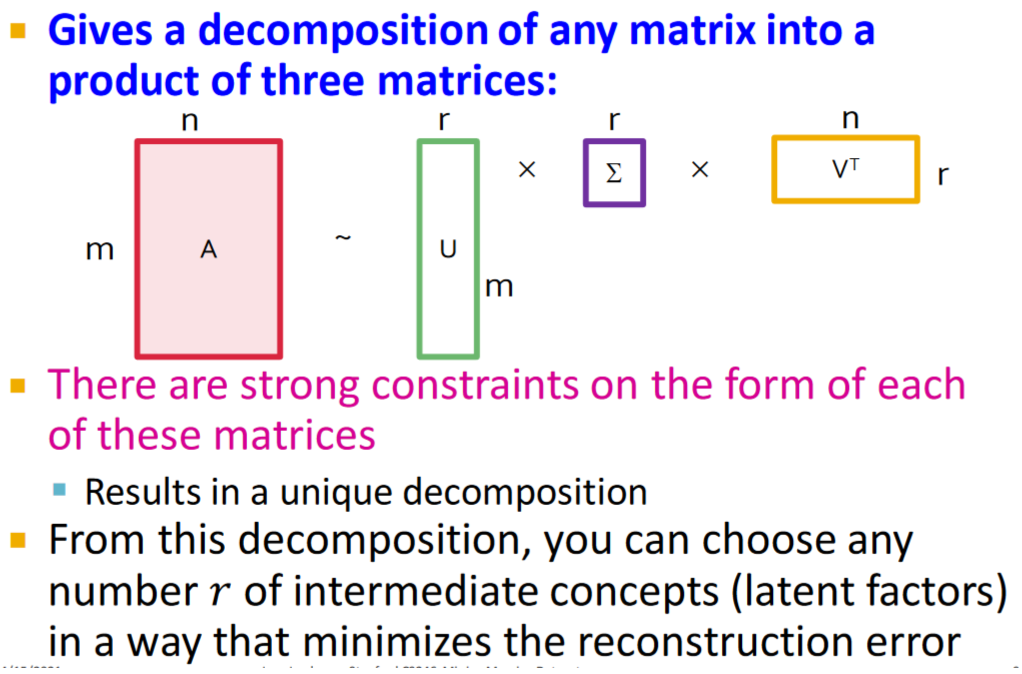
3. 对每个代表点移动一段距离，该距离是其位置到簇质心的距离乘以一个固定的比例。


Intuition:

- A large, dispersed cluster will have large moves from its boundary
- A small, dense cluster will have a little move.
- Favors a small, dense cluster that is near a larger dispersed cluster

上图给出了将代表点向质心方向移动的intuition。每个类有多于一个的代表点使得CURE可以适应非球形的几何形状。类的收缩或凝聚可以有助于控制孤立点的影响。因此，CURE对孤立点的处理更加健壮，而且能够识别非球形和大小变化比较大的类。针对大型数据库，CURE采用随机取样和划分两种方法组合：一个随机样本首先被划分，每个划分被部分聚类。

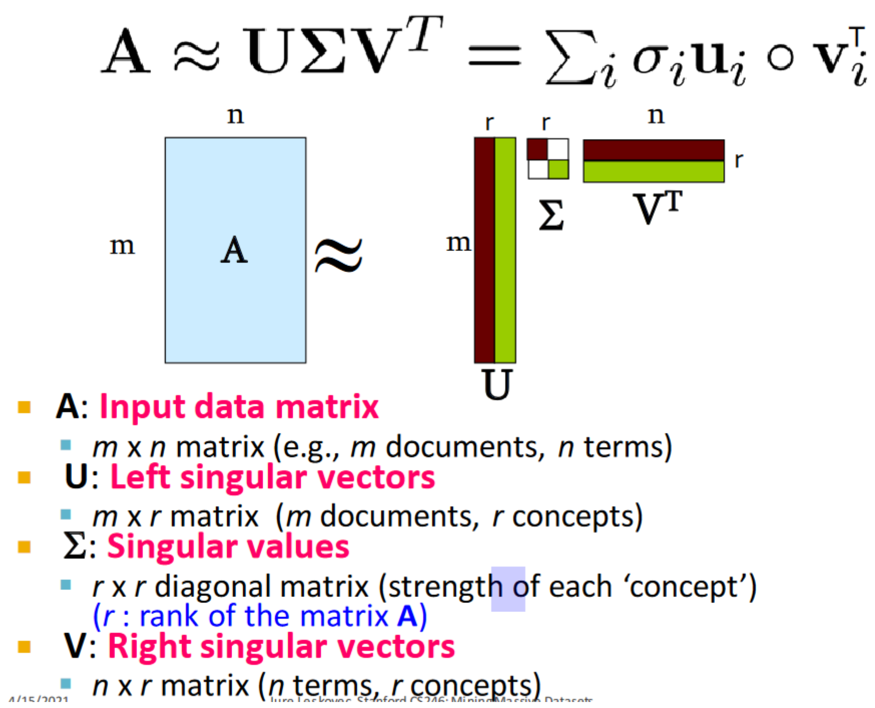
Dimension Reduction


- **Gives a decomposition of any matrix into a product of three matrices:**

$$\underset{m\times n}{A} \sim \underset{m\times r}{U} \times \underset{r\times r}{\Sigma} \times \underset{r\times n}{V^T}$$

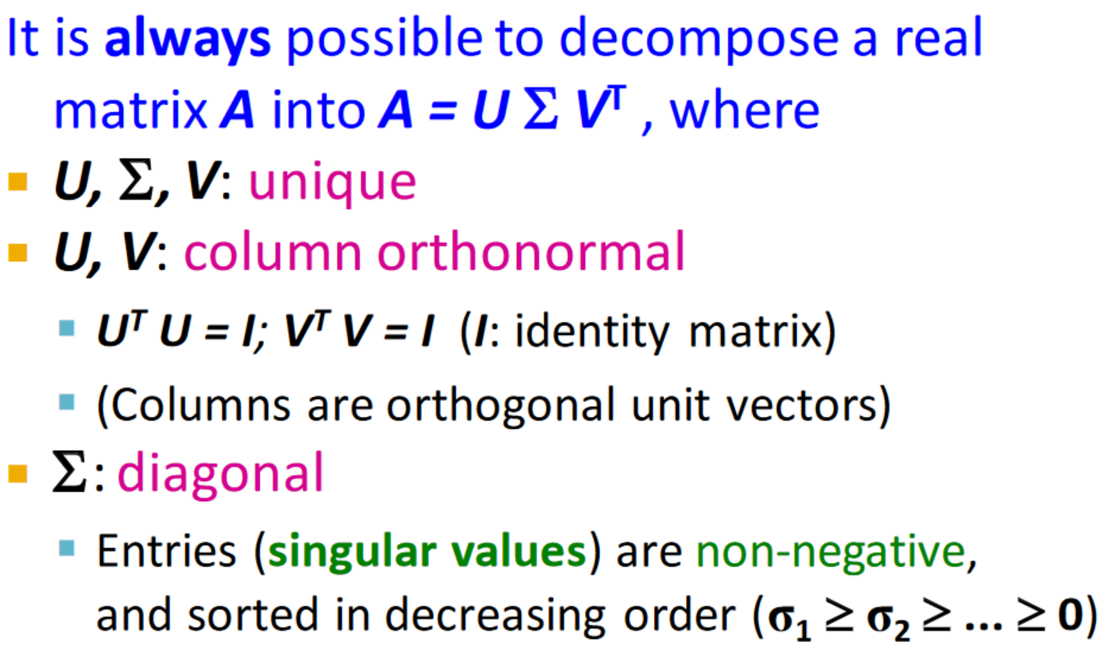
- There are strong constraints on the form of each of these matrices
  - Results in a unique decomposition
- From this decomposition, you can choose any number r of intermediate concepts (latent factors) in a way that minimizes the reconstruction error

直接给出SVD的定义。



$$\mathbf{A} \approx \mathbf{U}\Sigma\mathbf{V}^T = \sum_i \sigma_i \mathbf{u}_i \circ \mathbf{v}_i^T$$

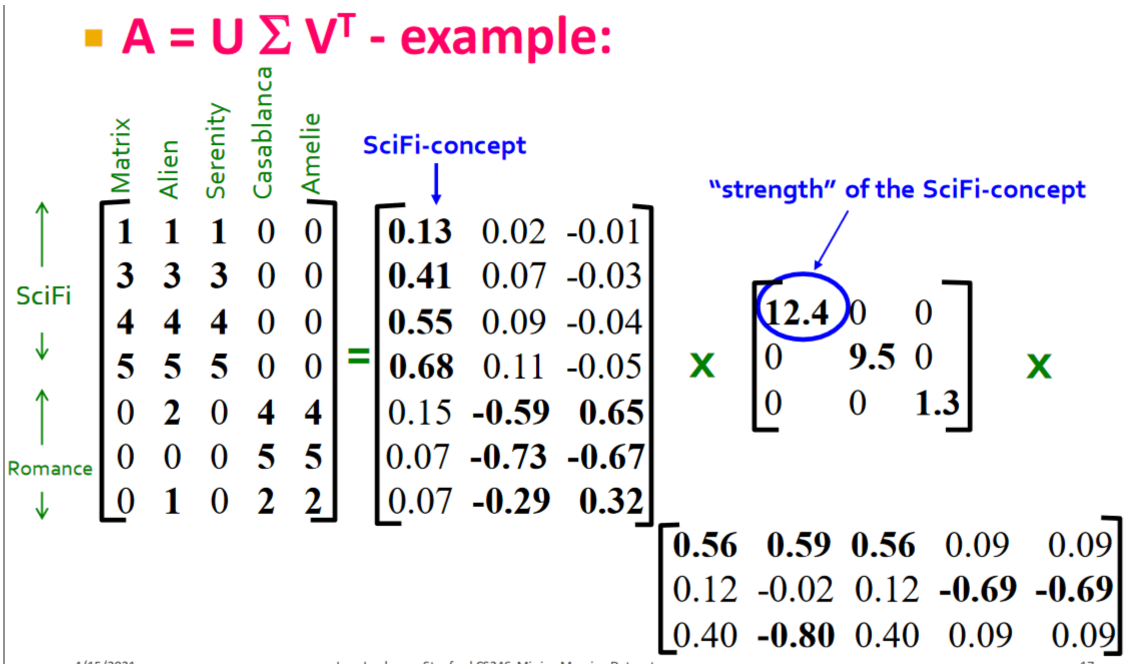
- **A**: **Input data matrix**
  - *m* x *n* matrix (e.g., *m* documents, *n* terms)
- **U**: **Left singular vectors**
  - *m* x *r* matrix (*m* documents, *r* concepts)
- Σ: **Singular values**
  - *r* x *r* diagonal matrix (strength of each 'concept')
    (*r* : rank of the matrix **A**)
- **V**: **Right singular vectors**
  - *n* x *r* matrix (*n* terms, *r* concepts)



It is **always** possible to decompose a real matrix ***A*** into ***A = U* Σ *V*^{T}** , where

- ***U*, Σ, *V***: unique
- ***U*, *V***: column orthonormal
  - ***U^T U = I*; *V^T V = I*** (***I***: identity matrix)
  - (Columns are orthogonal unit vectors)
- Σ: diagonal
  - Entries (**singular values**) are non-negative, and sorted in decreasing order ($\sigma_1 \geq \sigma_2 \geq \ldots \geq 0$)

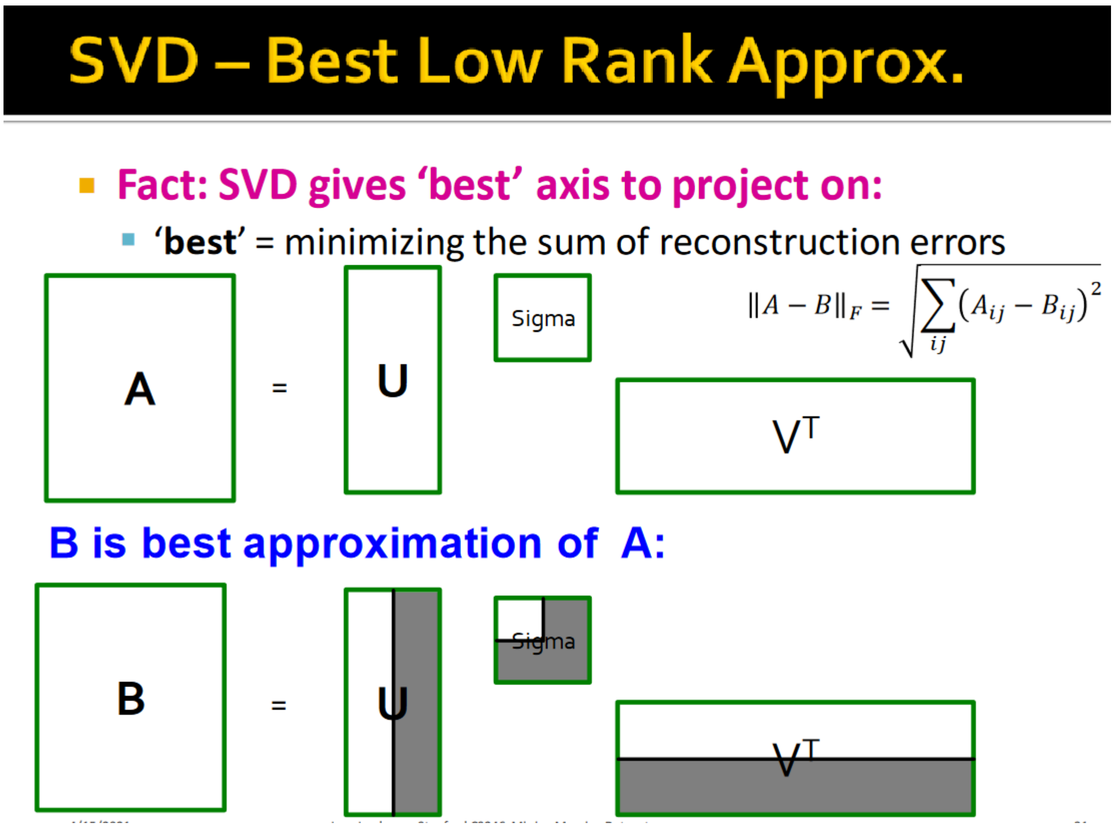
SVD分解实际上是将原矩阵A表示为latent factor内积的结果。如果将A矩阵视为user-item矩阵。那么得到的latent factor就是不同类别的item属性和不同性格的user属性。对角矩阵的元素大小可以被看做是"Strength"。



- $A = U \Sigma V^T$ - example:

$$\begin{array}{cc} & \begin{array}{ccccc} \text{Matrix} & \text{Alien} & \text{Serenity} & \text{Casablanca} & \text{Amelie} \end{array} \\ \begin{array}{c} \text{SciFi} \\ \\ \\ \\ \text{Romance} \\ \\ \\ \end{array} & \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix} \end{array} = \begin{bmatrix} \mathbf{0.13} & 0.02 & -0.01 \\ \mathbf{0.41} & 0.07 & -0.03 \\ \mathbf{0.55} & 0.09 & -0.04 \\ \mathbf{0.68} & 0.11 & -0.05 \\ 0.15 & \mathbf{-0.59} & \mathbf{0.65} \\ 0.07 & \mathbf{-0.73} & \mathbf{-0.67} \\ 0.07 & \mathbf{-0.29} & \mathbf{0.32} \end{bmatrix} \times \begin{bmatrix} \mathbf{12.4} & 0 & 0 \\ 0 & \mathbf{9.5} & 0 \\ 0 & 0 & \mathbf{1.3} \end{bmatrix} \times \begin{bmatrix} \mathbf{0.56} & \mathbf{0.59} & \mathbf{0.56} & 0.09 & 0.09 \\ 0.12 & -0.02 & 0.12 & \mathbf{-0.69} & \mathbf{-0.69} \\ 0.40 & \mathbf{-0.80} & 0.40 & 0.09 & 0.09 \end{bmatrix}$$

U矩阵为user-concept factor矩阵 V为item-concept facior矩阵 对角矩阵元素是concept的强度。



SVD – Best Low Rank Approx.

- **Fact: SVD gives 'best' axis to project on:**
  - '**best**' = minimizing the sum of reconstruction errors

$$\|A - B\|_F = \sqrt{\sum_{ij} (A_{ij} - B_{ij})^2}$$

B is best approximation of A:

How to Compute SVD

想要计算SVD，首先我们要找到计算对称方阵的特征值和主成分的方法。我们使用迭代的方式进行计算:

- 随机初始化特征向量x_0
- 计算$x_{k+1} = \frac{Mx_k}{||Mx_k||}$ $for\ k = 0, 1, \ldots$ M为对称方阵 分母表示F-norm
- 当x_i变化很小时停止迭代

- Once you have the principal eigenvector $\boldsymbol{x}$, you find its eigenvalue λ by $\lambda = \boldsymbol{x}^T M \boldsymbol{x}$.
  - In proof: We know $\boldsymbol{x}\lambda = M\boldsymbol{x}$ if λ is the eigenvalue; multiply both sides by $\boldsymbol{x}^T$ on the left.
  - Since $\boldsymbol{x}^T\boldsymbol{x} = 1$ we have $\lambda = \boldsymbol{x}^T M \boldsymbol{x}$

当我们找到特征向量x时，根据特征值定义我们有$\lambda = x^T Mx$ ，因此可以求出x对应的特征值。

- Eliminate the portion of the matrix M that can be generated by the first eigenpair, λ and $\boldsymbol{x}$:

$$M^* := M - \lambda\, x\, x^T$$

- Recursively find the principal eigenpair for M^*, eliminate the effect of that pair, and so on

在进行下一轮运算时，将矩阵M修改为$M - \lambda x x^T$ 。迭代计算新M矩阵对应的x和λ 。

我们得到了计算方阵特征向量和特征值的做法。对于SVD我们的做法相似：

- **Start by supposing $\boldsymbol{A} = \boldsymbol{U\Sigma V^T}$**
- $A^T = (U\Sigma V^T)^T = (V^T)^T\Sigma^T U^T = V\Sigma U^T$
  - Why? (1) Rule for transpose of a product; (2) the transpose of the transpose and the transpose of a diagonal matrix are both the identity functions
- $\boldsymbol{A^T A = V\Sigma U^T U \Sigma V^T = V\Sigma^2 V^T}$
  - Why? U is orthonormal, so $U^T U$ is an identity matrix
  - Also note that Σ^2 is a diagonal matrix whose i-th element is the square of the i-th element of Σ
- $\boldsymbol{A^T A V = V\Sigma^2 V^T V = V\Sigma^2}$
  - Why? V is also orthonormal

根据SVD的分解公式，我们将A转置后与他自身相乘得到$V\sum^2 V^T$， 右乘V后可以得到上图的最后一个等式。我们发现这就是$A^T A$矩阵的特征值定义，其中V的每一列都是特征向量，对应的特征值为对角矩阵的对应元素。因此求解A矩阵的SVD实际上可以通过求解$A^T A$的特征值和特征向量得到，问题转化为了我们已知的形式。注意这里得到的特征值需要开根号后才是A的奇异值，同时得到的奇异值矩阵的V。如果想要得到U矩阵，可以求解AA^T的特征向量来得到，推导与上述推导相同。

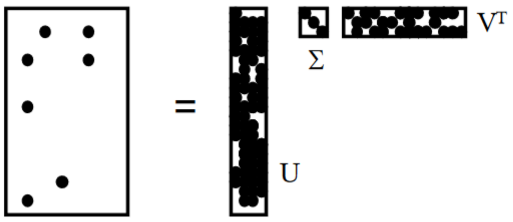
- **To compute the full SVD using specialized methods:**
  - **$O(nm^2)$** or **$O(n^2m)$** (whichever is less)
- **But:**
  - Less work, if we just want singular values
  - or if we want the first *k* singular vectors
  - or if the matrix is sparse
- **Implemented in** linear algebra packages like
  - LINPACK, Matlab, SPlus, Mathematica ...

SVD有如下的几个缺点：

+ **Optimal low-rank approximation** in terms of Frobenius norm

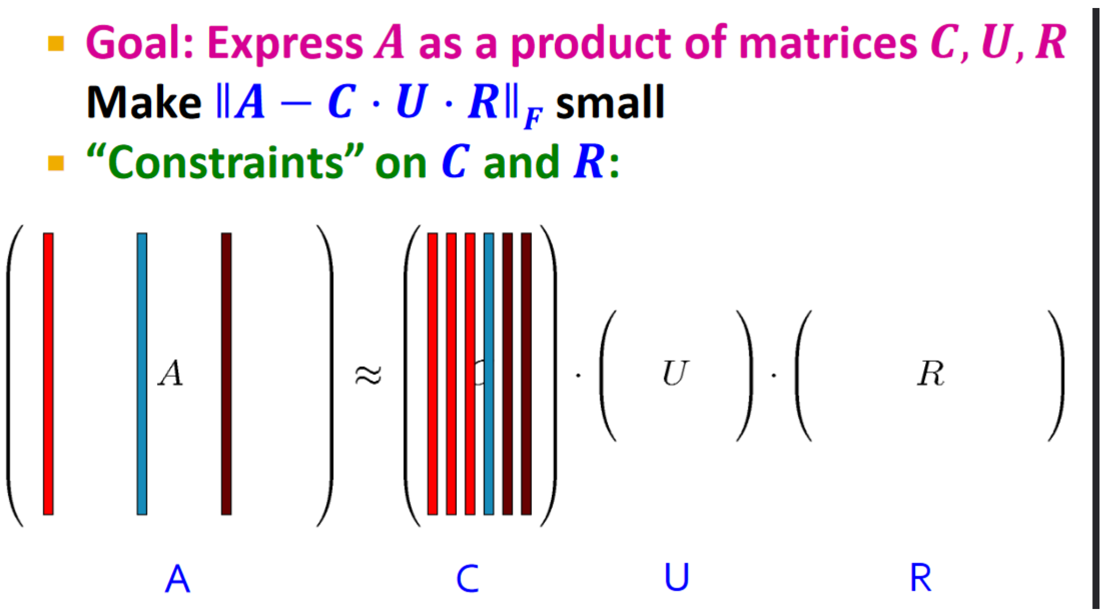
- **Interpretability problem:**
  - A singular vector specifies a linear combination of all input columns or rows

- **Lack of sparsity:**
  - Singular vectors are **dense!**



CUR Decomposition

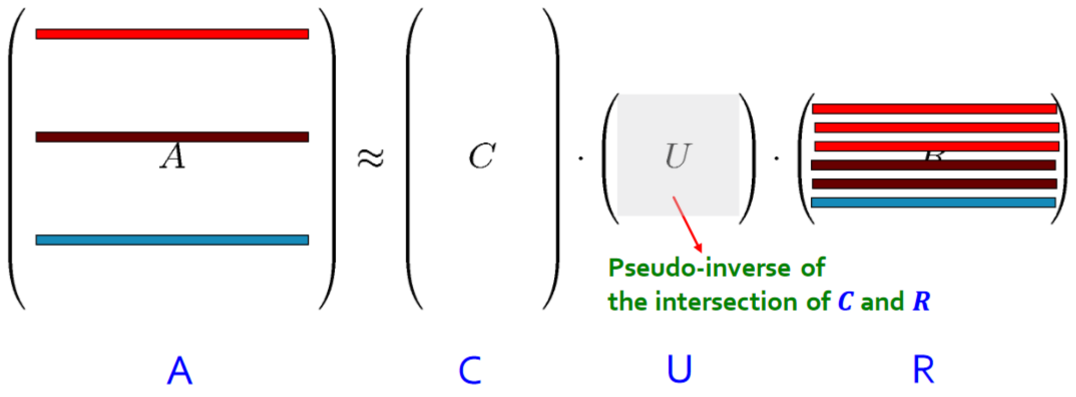
在实际场景中，我们希望进行分解的矩阵A可能是稀疏的，如user-item矩阵，用户可能只是购买了一小部分item。同时我们也希望分解得到的UV矩阵是稀疏的（更好的可解释性）。



- **Goal: Express A as a product of matrices C, U, R**

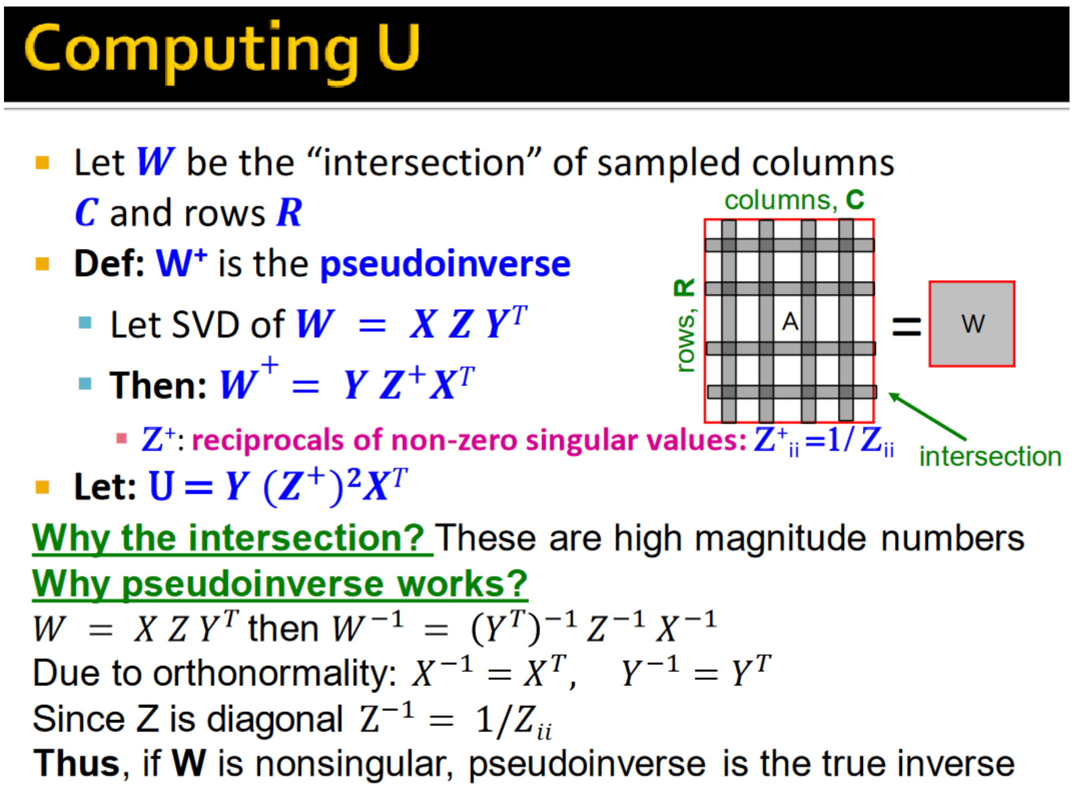
  Make $\|A - C \cdot U \cdot R\|_F$ small
- **"Constraints" on C and R:**

我们将A矩阵分解为CUR的矩阵乘积形式。同时将U矩阵定义为CR交集的伪逆矩阵。



$A \approx C \cdot U \cdot R$
Pseudo-inverse of the intersection of C and R
A
C
U
R

定义C矩阵为A矩阵的列抽样得到，R矩阵为A矩阵的行抽样得到。那么W矩阵为行列交点的元素构成的矩阵。



Computing U

- Let W be the "intersection" of sampled columns C and rows R
- **Def:** W^+ is the **pseudoinverse**
  - Let SVD of $W = X Z Y^T$
  - **Then:** $W^+ = Y Z^+ X^T$
    - Z^+: **reciprocals of non-zero singular values:** $Z^+_{ii} = 1/Z_{ii}$
- **Let:** $U = Y (Z^+)^2 X^T$

columns, C
rows, R
A
=
W
intersection

Why the intersection? These are high magnitude numbers

Why pseudoinverse works?

$W = X Z Y^T$ then $W^{-1} = (Y^T)^{-1} Z^{-1} X^{-1}$

Due to orthonormality: $X^{-1} = X^T, \quad Y^{-1} = Y^T$

Since Z is diagonal $Z^{-1} = 1/Z_{ii}$

Thus, if **W** is nonsingular, pseudoinverse is the true inverse

定义W^+为伪逆矩阵，$W^+ = YZ^+X^T$。

- To decrease the expected error between A and its decomposition, we must pick rows and columns in a nonuniform manner
- The **importance** of a row or column of A is the square of its Frobenius norm
  - That is, the sum of the squares of its elements.
- When picking rows and columns, the probabilities must be proportional to importance
- **Example:** [3,4,5] has importance 50, and [3,0,1] has importance 10, so pick the first 5 times as often as the second

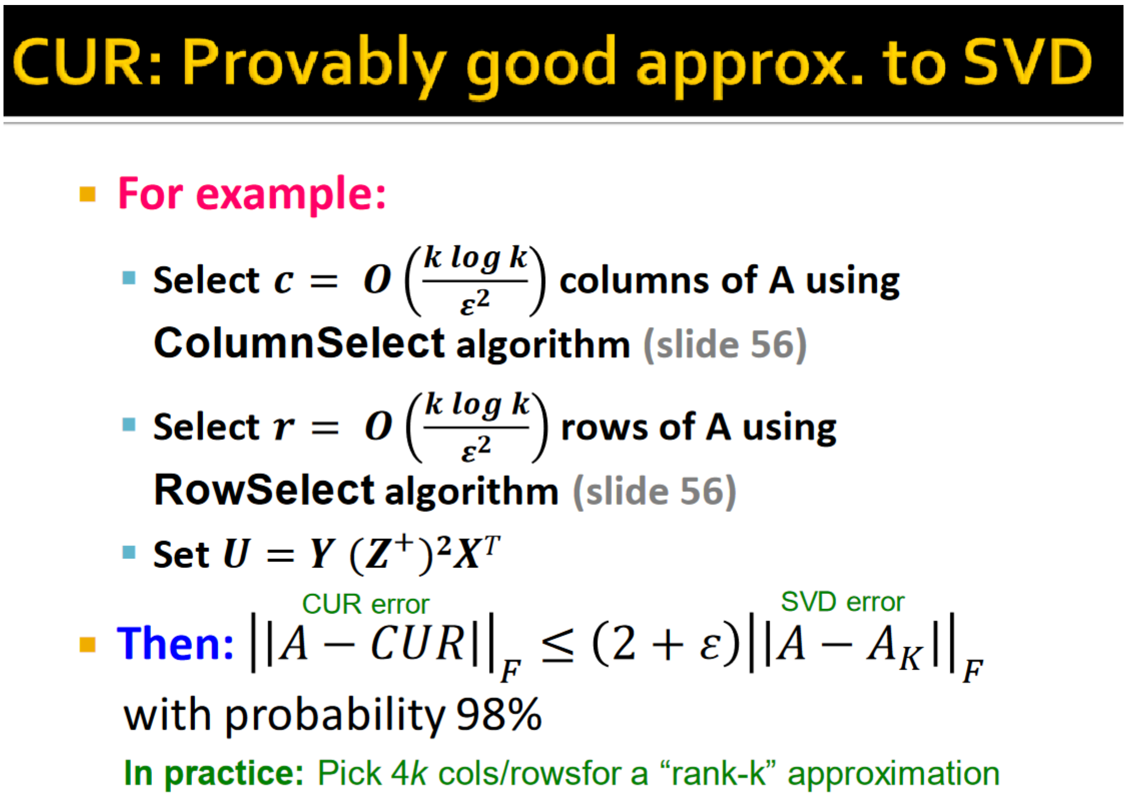
定义矩阵A的行或列的importance为对应行或列的元素平方和。以importance为权重对A矩阵进行C R矩阵的加权抽样。

- **Sampling columns (similarly for rows):**

```
Input: matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, sample size $c$
Output: $\mathbf{C}_d \in \mathbb{R}^{m\times c}$
    1. for $x = 1 : n$        [column distribution]
    2.     $P(x) = \sum_i \mathbf{A}(i,x)^2 / \sum_{i,j} \mathbf{A}(i,j)^2$
    3. for $i = 1 : c$        [sample columns]
    4.     Pick $j \in 1 : n$ based on distribution $P(x)$
    5.     Compute $\mathbf{C}_d(:,i) = \mathbf{A}(:,j)/\sqrt{cP(j)}$
```

Note this is a randomized algorithm, same column can be sampled more than once

注意对应的C R向量需要进行归一化操作。



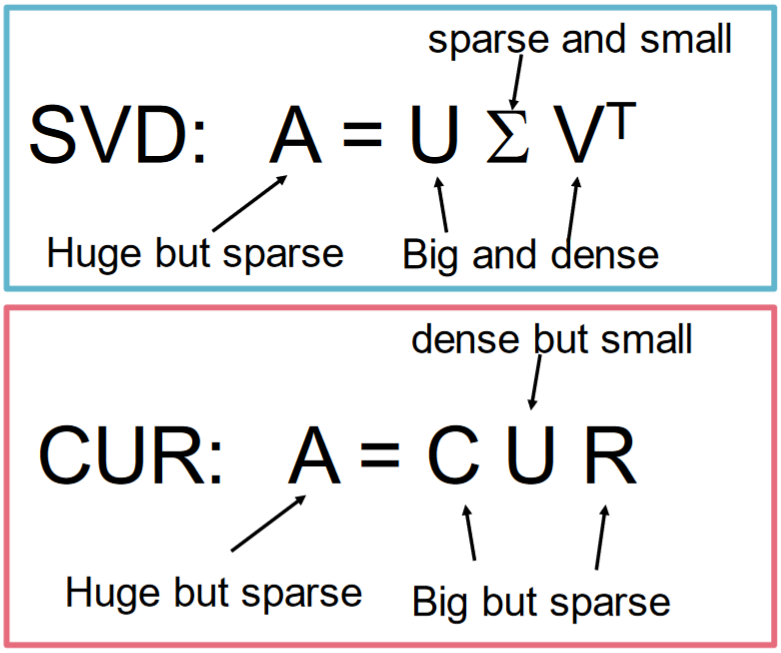
CUR: Provably good approx. to SVD

- **For example:**
  - **Select $c = O\left(\frac{k \log k}{\varepsilon^2}\right)$ columns of A using ColumnSelect algorithm** (slide 56)
  - **Select $r = O\left(\frac{k \log k}{\varepsilon^2}\right)$ rows of A using RowSelect algorithm** (slide 56)
  - **Set $U = Y\,(Z^+)^2 X^T$**
- **Then:** $\underbrace{\left|\left|A - CUR\right|\right|_F}_{\text{CUR error}} \le (2+\varepsilon)\underbrace{\left|\left|A - A_K\right|\right|_F}_{\text{SVD error}}$

  with probability 98%

  In practice: Pick $4k$ cols/rowsfor a “rank-k” approximation

CUR分解对于A矩阵的误差上界由上图给出。



对比SVD和CUR分解，由于CR矩阵是直接从较为稀疏的A矩阵中直接采样得到，所以CR矩阵仍然是稀疏的。