Recommender Systems

Dimensionality Reduction

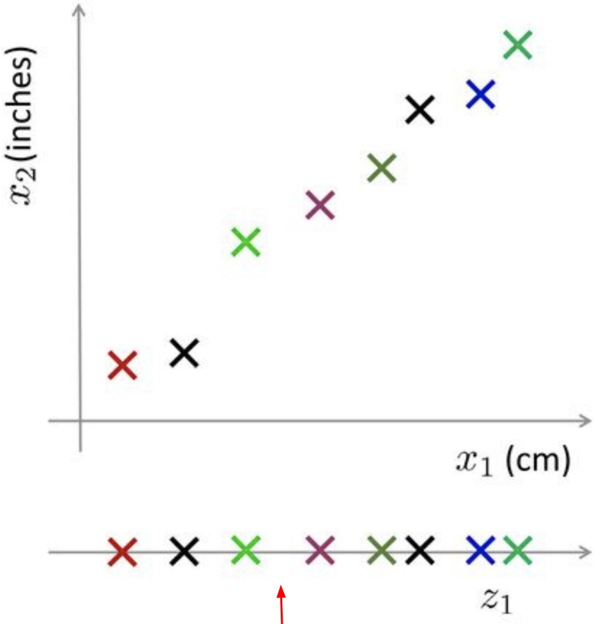
E0 259: Data Analytics
Lecture 2

User-Item Matrix

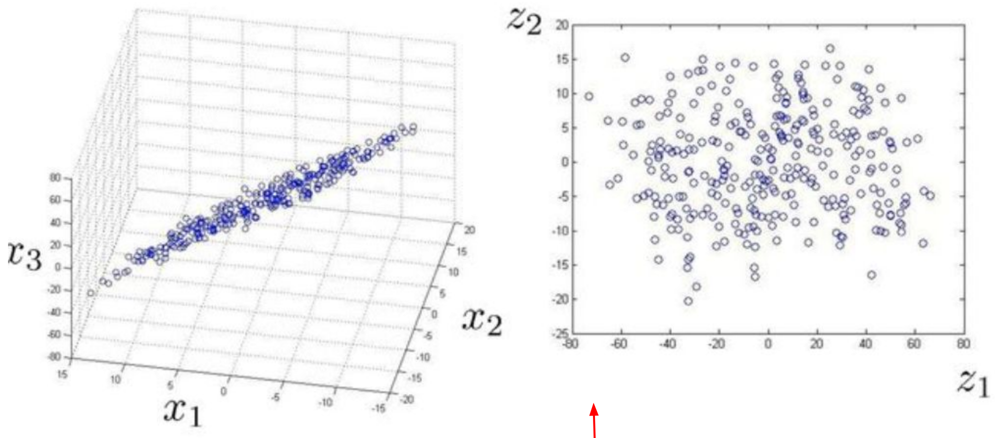
- Step 1: Number of rows and columns can be order of millions to billions! How can we reduce this dimension for efficient storage and computation?
- Step 2: How do we find nearest neighbors efficiently.

We will first focus on Step 1

Dimensionality Reduction



1D Enough to represent data



2D Enough to represent data

In large data sets orders of magnitude reduction in dimensions possible

Matrix Rank

Movie/User	Iron Man	Dr. Strange	Thor	Superman	Batman
Alice	4	4	4		
Bob	3	3	3		
Charlie				2	2
Doug				5	5

- Only 2 dimensions needed to represent this
- [1,1,1,0,0] and [0,0,0,1,1].
- Any row in matrix is linear combination of these 2 (extreme case)
- Rank of matrix is 2!

What If?

Movie/User	Iron Man	Dr. Strange	Thor	Superman	Batman
Alice	4	4	4		1
Bob	3	3	3		
Charlie				2	2
Doug				5	5

- [1,1,1,0,0] and [0,0,0,1,1] cannot represent this!
- But … if we did, error is small!
- Idea behind dimensionality reduction

How Does Reducing Dimensions Help?

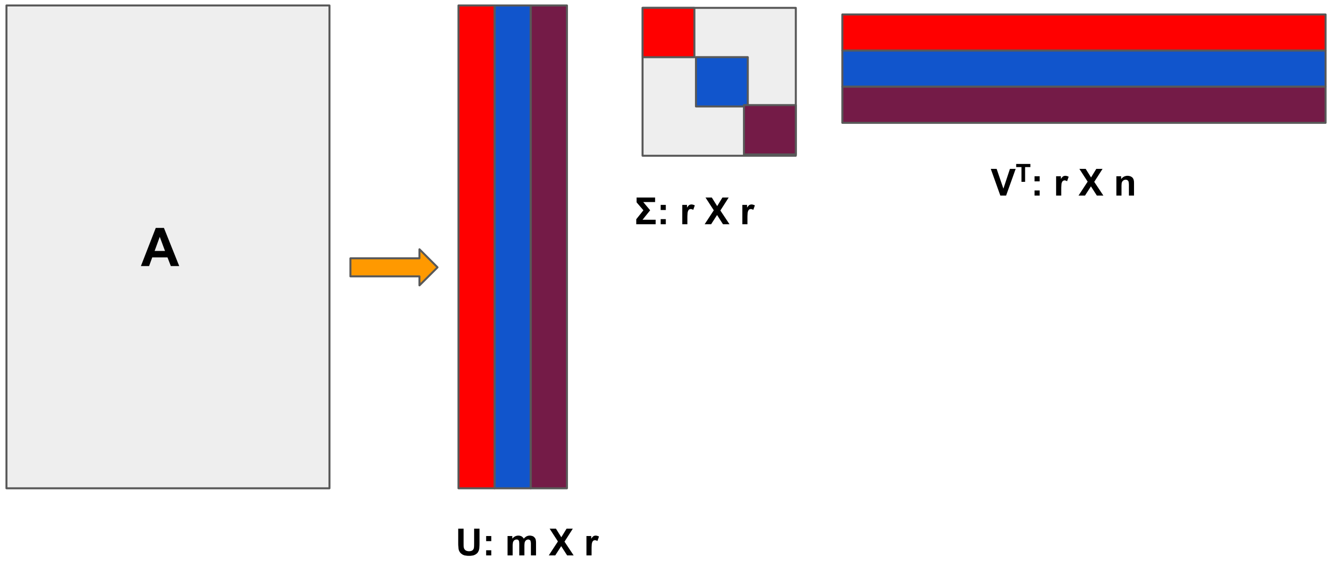
- Remove redundant features
- Discover hidden correlations
  - If 2 features are exactly correlated, can be represented with one less dimension
- Lower storage and compute costs to find similar items/users

Singular Value Decomposition

$$\mathbf{A}_{\mathbf{mXn}} = \mathbf{U}_{\mathbf{mXr}} \mathbf{\Sigma}_{\mathbf{rXr}} \mathbf{V}^{\mathbf{T}}_{\mathbf{rXn}}$$

- **A :** m X n matrix - users to items ratings matrix
- **U :** m X r matrix - Left Singular Vector matrix: users to ‘latent factors’ or perhaps genres matrix
  - Represents user affinity to latent factors
- **Σ :** r X r matrix - Singular Values: strength of ‘latent factors’ or perhaps genres.
  - How strong or important each ‘latent factor’ is
- **V:** r X n matrix - Right Singular Vector matrix: ‘latent factors’ to items matrix
  - Represents mixture of ‘latent factors’ in item

SVD - Visually



SVD (contd.)

- We can apply SVD on any real matrix
- **U, Σ** and **V are** unique
- U^TU and $V^TV = I$ (identity matrix)
    - For any two columns in U, $u_i^Tu_j = 0$, if i and j different
    - For any two columns in V, $v_i^Tv_j = 0$, if i and j different
- **Σ** is a diagonal matrix
    - Entries are positive
    - Entries are in decreasing order

User/Movie	Iron Man	Dr. Strange	Thor	Superman	Batman
Alice	4	4	4		1
Bob	3	3	3		
Charlie				2	2
Doug				5	5

Marvel DC

User/LF	LF1	LF2	LF3
Alice	**0.79**	-0.15	0.59
Bob	**0.57**	-0.18	-0.8
Charlie	0.08	**0.36**	-0.02
Doug	0.21	**0.9**	-0.05

U: m X r

LF/LF	LF1	LF2	LF3
LF1	**8.75**	0	0
LF2	0	**7.56**	0
LF3	0	0	0.42

Σ: r X r

Can ignore LF3

LF/Movie	Iron Man	Dr. Strange	Thor	Superman	Batman
LF1	**0.56**	**0.56**	**0.56**	0.13	0.23
LF2	-0.15	-0.15	-0.15	**0.69**	**0.67**
LF3	-0.04	-0.04	-0.04	-0.7	0.7

V^T: r X n

User/LF	LF1	LF2	LF3
Alice	**0.79**	-0.15	0.59
Bob	**0.57**	-0.18	-0.8
Charlie	0.08	**0.36**	-0.02
Doug	0.21	**0.9**	-0.05

U: m X r

LF/LF	LF1	LF2	LF3
LF1	**8.75**	0	0
LF2	0	**7.56**	0
LF3	0	0	~~0.42~~

Σ: r X r

LF/Movie	Iron Man	Dr. Strange	Thor	Superman	Batman
LF1	**0.56**	**0.56**	**0.56**	0.13	0.23
LF2	-0.15	-0.15	-0.15	**0.69**	**0.67**
LF3	-0.04	-0.04	-0.04	-0.7	0.7

V^T: r X n

User/LF	LF1	LF2
Alice	**0.79**	-0.15
Bob	**0.57**	-0.18
Charlie	0.08	**0.36**
Doug	0.21	**0.9**

LF/LF	LF1	LF2
LF1	**8.75**	0
LF2	0	**7.56**

LF/Movie	Iron Man	Dr. Strange	Thor	Superman	Batman
LF1	**0.56**	**0.56**	**0.56**	0.13	0.23
LF2	-0.15	-0.15	-0.15	**0.69**	**0.67**

User/Movie	Iron Man	Dr. Strange	Thor	Superman	Batman
Alice	4	4	4		1
Bob	3	3	3		
Charlie				2	2
Doug				5	5

User/Movie	Iron Man	Dr. Strange	Thor	Superman	Batman
Alice	4.04	4.04	4.04		0.83
Bob	2.99	2.99	2.99	-0.29	0.23
Charlie	-0.01	-0.01	-0.01	1.97	1.98
Doug	0.008	0.008	0.008	4.93	4.98

If you want to minimize mean square error, start removing from smallest singular value

Best Low Rank Approximation

Theorem. *Let* $\mathbf{A} = \mathbf{U\Sigma V^T}$*, where* $\Sigma : \sigma_1 \geq \sigma_2, \ldots,$ *and rank*$(\mathbf{A}) = \mathbf{r}$*, then* $\mathbf{A} = \mathbf{U_k \Sigma_k V_k^T}$*, is a best rank k approximation to* $\mathbf{A}$ *where,* $S_k = \sigma_1, \ldots, \sigma_k$*,* U_k *is first k columns of* U *and* V_k *is the first k columns of* V*.*

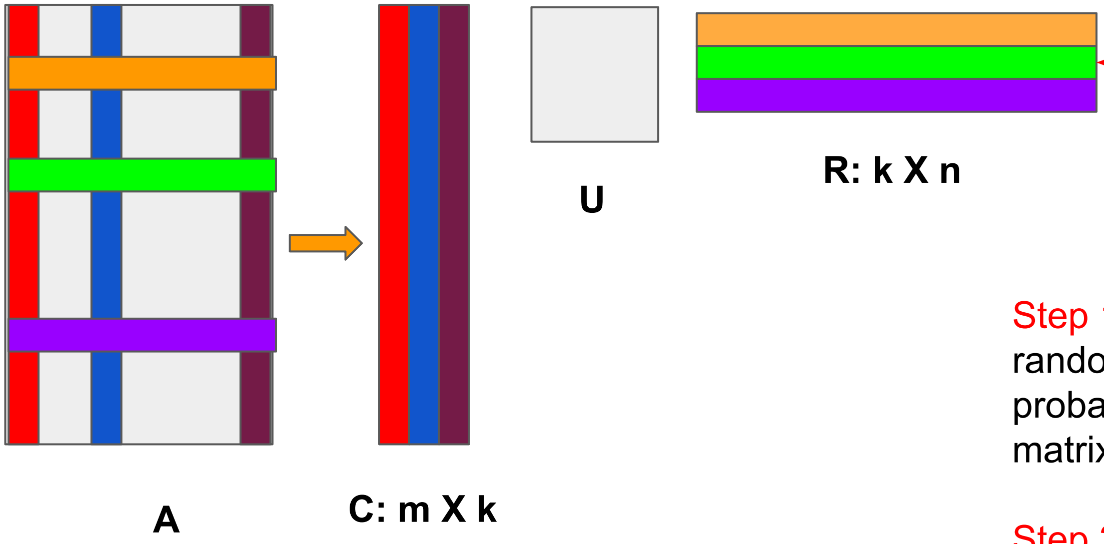
Best implies that $\mathbf{B} = \mathbf{min_B} \|\mathbf{A_B}\|_\mathbf{F}$ *where rank*$(\mathbf{B}) = \mathbf{k}$

In practice try to capture 80 - 90% of the original variance. Gives good dimensionality reduction
SVD Complexity - $O(n^2m)$ or $O(m^2n)$

CUR Decomposition

- Complexity of computing SVD $O(n^2m)$ or $O(m^2n)$
- Can we do better?

CUR Decomposition



Pick k random rows from A

Pick k random columns from A

Step 1: Pick k columns and rows at random with some carefully chosen probabilities. This will be C and R matrix

Step 2: Look at intersecting rows and columns C and R. This will be small matrix. Do SVD on it.

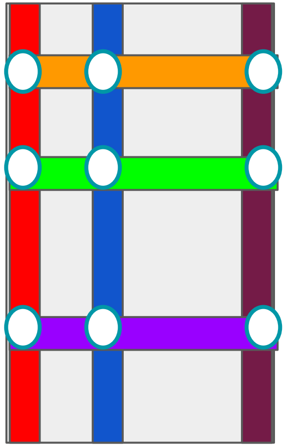
Step 1: Picking *k* Columns

- Assume you want to reduce dimension to k
- Total energy in matrix $A = \sum_{i,j} A_{ij}^2$
- Energy in column $j = \sum_i A_{ij}^2$
- Probability of picking column j, $P_j = \frac{\sum_i A_{ij}^2}{\sum_{i,j} A_{ij}^2}$
- If column j is picked, normalize values of column j to $\frac{A_{ij}}{\sqrt{(kP_j)}}$

Same trick for picking rows

Step 2: Figuring out ***U***

- Pick intersecting rows and columns of C and R. Let this be W
- Do SVD on $W = X\Sigma Y^T$
- U= pseudo inverse of W, $U = Y\Sigma^+ X^T$



Step 2: Figuring out ***U*** (contd.)

- i.e., Σ^+ is reciprocal of non zero singular values
- $W^{-1} = (Y^T)^{-1}\Sigma^{-1}X^{-1}$
- But $X^T X = I$ and $Y^Y Y = I$
- $\Sigma^+_{ii} = \frac{1}{\Sigma_{ii}}$ Can only use non zero values.
- Therefore $U = Y\Sigma^+ X^T$

Theorem Drineas et. al

- Let A_k be k rank SVD approximation of A

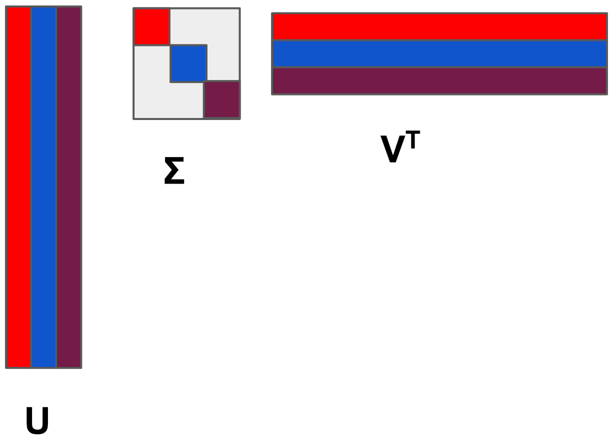
Theorem. *CUR is $O(nm)$ and achieves*

- $\|A - CUR\| \leq \|A - A_k\| + \epsilon\|A\| \;\; w.p \geq 1 - \delta$
- *by picking* $O(k\frac{\log(1/\delta)}{\epsilon^2})$ *columns and*
- $O(k^2\frac{\log(^3 1/\delta)}{\epsilon^6})$

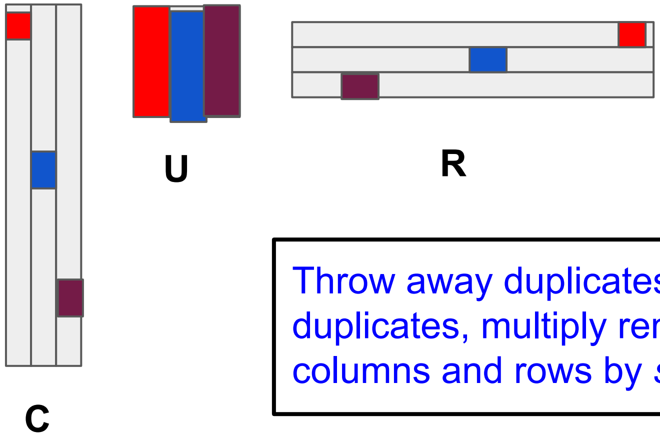
In practice pick 4k rows and columns

CUR Method vs SVD

- SVD -
  - U and V dense
  - Σ sparse
- m and n can be billions
  - Storage explodes
- High computational complexity

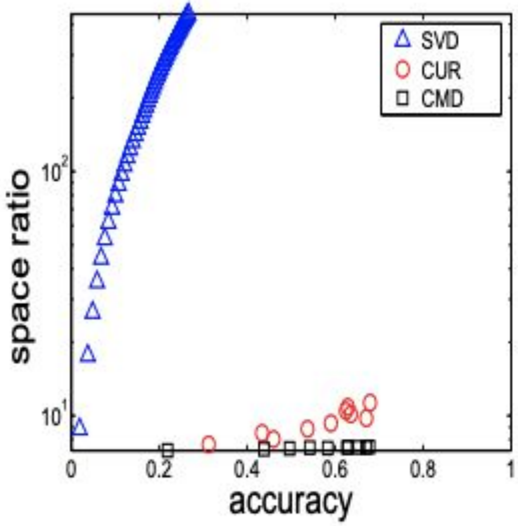


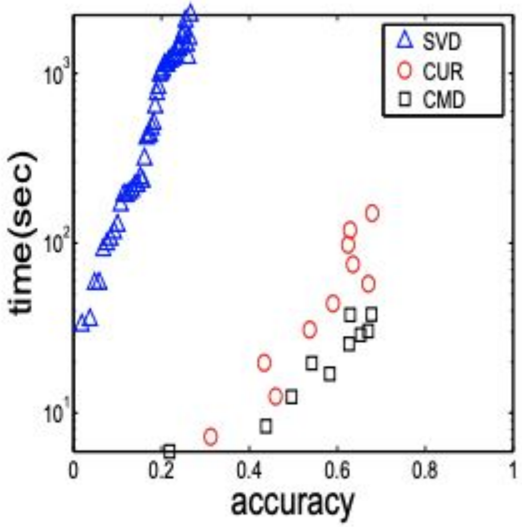
- CUR -
  - C and R sparse,
  - U dense
- Storage small
- Low computational time
- Can pick duplicates - tricks around this



Throw away duplicates. If *d* duplicates, multiply remaining columns and rows by *sqrt(d)*

DBLP Dataset





Falustos et. al, 2007

Latent Factor Models

- Problem with SVD, CUR etc.
  - Treats unrated items as 0, rather than trying to predict them well.
  - Each time new users, items etc come in, need to reurn!

User/Movie	Iron Man	Dr. Strange	Thor	Superman	Batman
Alice	4	4	4		1
Bob	3	3	3		
Charlie				2	2
Doug				5	5

User/Movie	Iron Man	Dr. Strange	Thor	Superman	Batman
Alice	4.04	4.04	4.04		0.83
Bob	2.99	2.99	2.99	-0.29	0.23
Charlie	-0.01	-0.01	-0.01	1.97	1.98
Doug	0.008	0.008	0.008	4.93	4.98

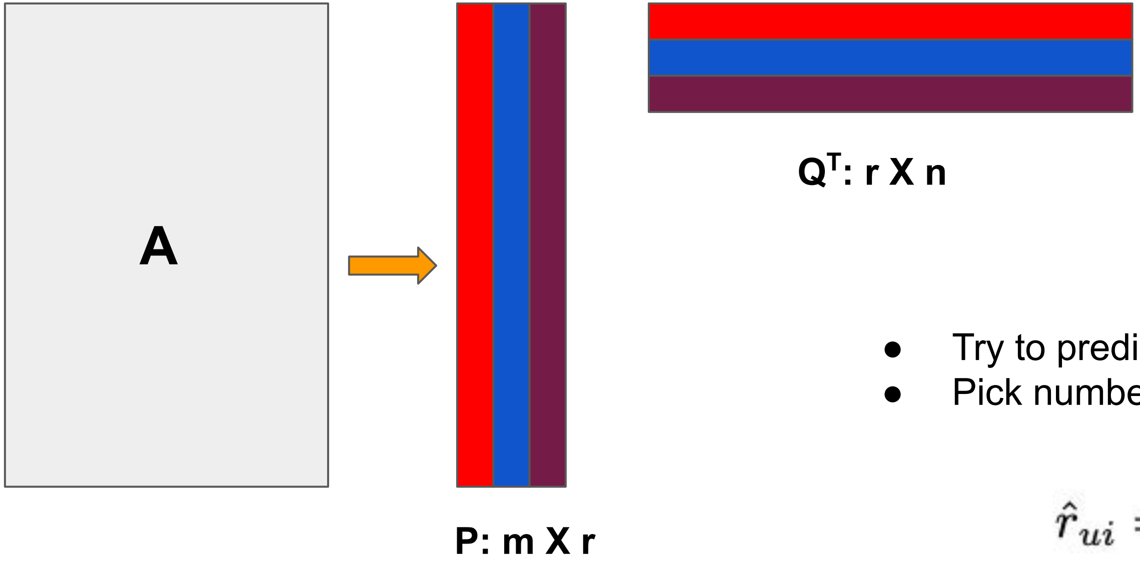
Take a Machine Learning Approach

User/Movie	Iron Man	Dr. Strange	Thor	Superman	Batman
Alice	4	4	4		1
Bob	3	3	3		
Charlie				2	2
Doug				5	5

Training Data

Test Data

Latent Factor Model

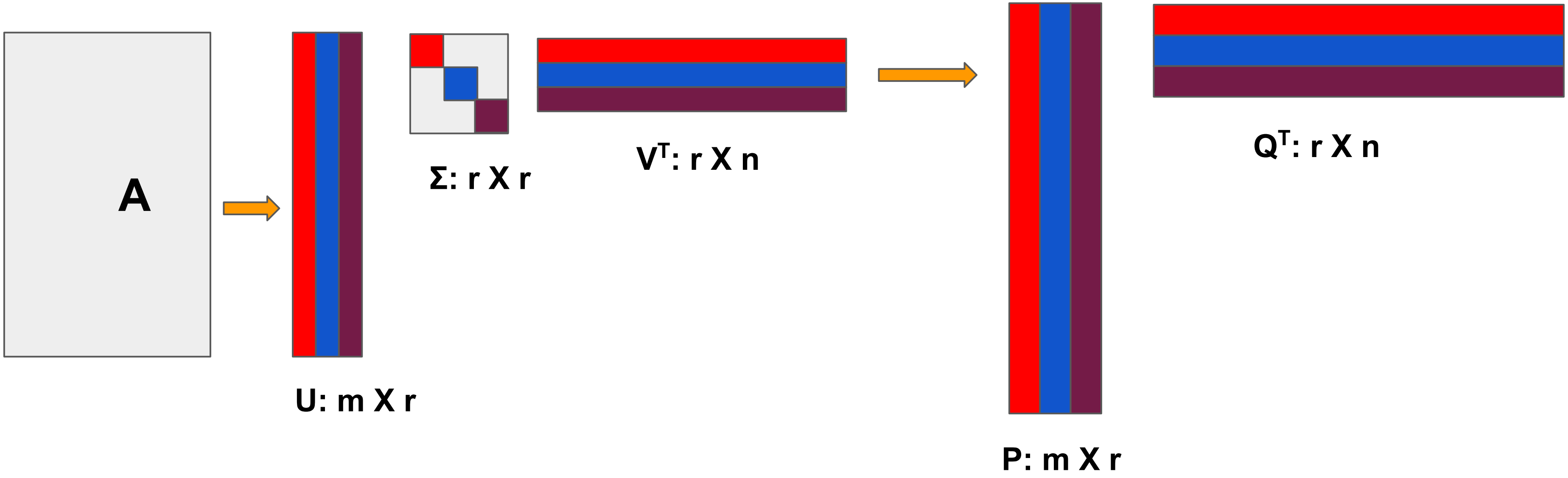


- Try to predict accurately on known ratings
- Pick number of factors *k* apriori

$$\hat{r}_{ui} = \mathbf{p_u}\mathbf{q_i^T}$$

$$= \sum_k q_{uk} p_{ik}$$

Relating back to SVD



$U = P$, $\quad$ **$\Sigma V^T = Q^T$**

- For a given *k*, SVD minimizes RMSE
- Latent Factor Model Loss Function
  - minimize Mean Square Error
  - Use only cells with ratings

Latent Factor Model - Loss Function

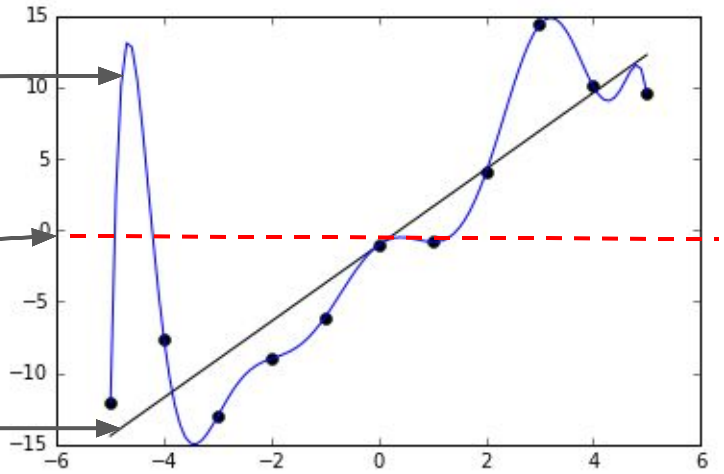
$$\min_{\mathbf{P},\mathbf{Q}} \sum_{(\mathbf{u},\mathbf{i})\in \mathbf{A}} (\mathbf{r_{ui}} - \mathbf{q_u}.\mathbf{p_i^T})^2$$

Minimize only over entries with user ratings!

Avoid Overfitting and apply Gradient Descent

$$\min_{\mathbf{P},\mathbf{Q}} \sum_{(\mathbf{u},\mathbf{i})\in \mathbf{A}} (\mathbf{r_{ui}} - \mathbf{q_u}.\mathbf{p_i^T})^2 + \lambda(\sum_{\mathbf{u}} \|\mathbf{q_u}\|^2 + \sum_{\mathbf{i}} \|\mathbf{p_i}\|^2)$$

- When λ small, then overfitting!
- When λ large, underfitting!
- λ is hyperparameter to tune for best fit



Use Gradient Descent to Find best Q and P

$$\mathbf{P_{t+1}} = \mathbf{P_t} - \epsilon \nabla \mathbf{P_t}$$

$$\mathbf{Q_{t+1}} = \mathbf{Q_t} - \epsilon \nabla \mathbf{Q_t}$$

$$\nabla q_{uk} = \sum_{ui} -2(r_{ui} - q_u p_i^T) p_{ik} + 2\lambda q_{uk}$$

$$\nabla p_{ik} = \sum_{ui} -2(r_{ui} - q_u p_i^T) q_{uk} + 2\lambda p_{ik}$$

- Can iteratively converge to good solution
- Can deal with dynamics of new user ratings
- Alternate Least Squares!

But after all this, we still have…

- Most user feedback is implicit (no explicit ratings) and …
- The Cold Start Problem