

Singular Value Decomposition, and Application to Recommender Systems

CSE 4309 – Machine Learning
Vassilis Athitsos
Computer Science and Engineering Department
University of Texas at Arlington

Recommendation Systems – Netflix Version

- Netflix users watch movies, and can rate each movie with an integer from 1 to 5.
- Netflix tries to **recommend** movies to users.
- To produce recommendations for a user U , one could follow a process like this:
 - For every movie M available on Netflix (except movies that user U has already watched), predict (i.e., guess) the rating that user U would give to movie M .
 - This guess is based on ratings that user U has provided in the past.
 - Recommend to user U the movies for which the predicted rating is the highest.

The Netflix Prize

- Back in 2006, Netflix announced a \$1 million prize for the first team to produce a recommendation algorithm whose accuracy exceeded a certain threshold.
- To help researchers, Netflix made available a large training set from its user data:
 - 470,189 users.
 - 17,770 movies.
 - About 100 million ratings.
- The prize was finally claimed in 2009.

Recommendation Systems – Amazon Version

- Amazon users buy products, and occasionally rate those products.
- Amazon tries to **recommend** products to users.
- To produce recommendations for a user U , one could follow a process like this:
 - For every product M available on Amazon (except products that user U has already purchased), predict (i.e., guess) the rating that user U would give to product M .
 - This guess is based on ratings that user U has provided in the past.
 - Recommend to user U the products for which the predicted rating is the highest.

Recommendation Systems – Amazon Version 2 (No Ratings)

- Amazon users buy products.
- Amazon tries to **recommend** products to users.
- To produce recommendations for a user U , one could follow a process like this:
 - For every product M available on Amazon (except products that user U has already purchased), predict (i.e., guess) a score of how much user U would want to buy product M .
 - This guess is based on products that user U has bought in the past.
 - Recommend to user U the products for which the predicted score is the highest.

A Disclaimer

- There are many different ways in which these recommendation problems can be defined and solved.
- Our goal in this class is to understand some basic methods.
- The way that actual companies define and solve these problems is usually more complicated.

Modeling Ratings

- We want a function $F(U, M)$ that predicts the rating that a user U would give to a product M .
- Our training data can be represented as a matrix A , where:
 - Every row corresponds to a user.
 - Every column corresponds to a product.
 - Value $A_{U,M}$ at row U and column M contains the rating that user U has given to product M , **if user U has actually rated product M .**
 - If U has not rated M , value $A_{U,M}$ is a **missing value**.
 - In other words, somehow we must tell the system that there is no valid value at row U and column M .

Modeling Ratings – An Example

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6	Movie 7	Movie 8	Movie 9
User 1	0	1	1	1	0	0	1	0	0
User 2	1	0	1	0	0	1	0	0	0
User 3	0	0	0	0	1	0	0	0	1
User 4	0	1	0	1	0	0	0	1	0
User 5	1	0	0	0	1	0	0	0	1
User 6	0	0	1	0	0	0	1	0	1
User 7	1	0	0	0	0	1	0	0	0

- Value $A_{U,M}$ at row U and column M contains the rating that user U has given to product M .
- Here is a simple case, with users and movies.
 - The value is 1 if the user has watched the movie, 0 otherwise.
 - There are no missing values.
 - Goal: train a system to recommend movies to the user.

Modeling Ratings – An Example

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6	Movie 7	Movie 8	Movie 9
User 1	0	1	1	1	0	0	1	0	0
User 2	1	0	1	0	0	1	0	0	0
User 3	0	0	0	0	1	0	0	0	1
User 4	0	1	0	1	0	0	0	1	0
User 5	1	0	0	0	1	0	0	0	1
User 6	0	0	1	0	0	0	1	0	1
User 7	1	0	0	0	0	1	0	0	0

- How can we make any predictions?
 - If a user has not watched a movie, how can we estimate how much the user would be interested?

Modeling Ratings – An Example

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6	Movie 7	Movie 8	Movie 9
User 1	0	1	1	1	0	0	1	0	0
User 2	1	0	1	0	0	1	0	0	0
User 3	0	0	0	0	1	0	0	0	1
User 4	0	1	0	1	0	0	0	1	0
User 5	1	0	0	0	1	0	0	0	1
User 6	0	0	1	0	0	0	1	0	1
User 7	1	0	0	0	0	1	0	0	0

- Each user prefers specific movies with specific characteristics.
- Each movie has certain characteristics.
- If we knew what characteristics each user likes, and what characteristics each movie has, we would have enough information to make predictions.
- However, we know neither what each user likes, nor what the characteristics of the movies are.

Modeling Ratings – An Example

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6	Movie 7	Movie 8	Movie 9
User 1	0	1	1	1	0	0	1	0	0
User 2	1	0	1	0	0	1	0	0	0
User 3	0	0	0	0	1	0	0	0	1
User 4	0	1	0	1	0	0	0	1	0
User 5	1	0	0	0	1	0	0	0	1
User 6	0	0	1	0	0	0	1	0	1
User 7	1	0	0	0	0	1	0	0	0

- There is another source of information.
- Some users have similar preferences.
 - For example, User 2 and User 7 agree on 8 of the nine values.
- Intuitively, some users like similar things in movies. We may not know what the users like, but we can tell they like similar things.

Modeling Ratings – An Example

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6	Movie 7	Movie 8	Movie 9
User 1	0	1	1	1	0	0	1	0	0
User 2	1	0	1	0	0	1	0	0	0
User 3	0	0	0	0	1	0	0	0	1
User 4	0	1	0	1	0	0	0	1	0
User 5	1	0	0	0	1	0	0	0	1
User 6	0	0	1	0	0	0	1	0	1
User 7	1	0	0	0	0	1	0	0	0

- Similarly, some movies have similar scores.
 - For example, Movie 1 and Movie 5 agree in six of their seven scores.
- So, some movies have things in common. We may not know what those things are, but we can tell that there are similarities.

Singular Value Decomposition

- Singular Value Decomposition (SVD) is a **matrix factorization method**.
- Given as input an $m \times n$ matrix A , SVD computes matrices U, S, V such that:
 - $A = USV^T$.
 - U is an $m \times m$ square matrix.
 - S is an $m \times n$ diagonal matrix (only entries $S_{i,i}$ are non-zero).
 - V is an $n \times n$ matrix.
- In a bit, we will discuss how to compute matrices U, S, V .
- First, we need to understand: How can SVD be useful in our problem?

- Consider $A_{2,3}$.
- It shows whether User 2 has watched Movie 3.
- How do we compute $A_{2,3}$ using $\mathbf{U}, \mathbf{S}, \mathbf{V}$?
- Let \mathbf{u}_2 be the second row of \mathbf{U} .
- Let \mathbf{v}_3 be the third row of \mathbf{V} .
- Then, $A_{2,3} = \mathbf{u}_2 \mathbf{S}(\mathbf{v}_3)^T =$

$$\sum_{d=1}^7 U_{2,d} S_{d,d} V_{3,d}$$

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6	Movie 7	Movie 8	Movie 9
User 1	0	1	1	1	0	0	1	0	0
User 2	1	0	1	0	0	1	0	0	0
User 3	0	0	0	0	1	0	0	0	1
User 4	0	1	0	1	0	0	0	1	0
User 5	1	0	0	0	1	0	0	0	1
User 6	0	0	1	0	0	0	1	0	1
User 7	1	0	0	0	0	1	0	0	0

$$\begin{bmatrix} \vdots & \ddots & \vdots \\ \vdots & \ddots & \vdots \\ \vdots & \ddots & \vdots \end{bmatrix} \times \begin{bmatrix} \vdots & \ddots & \vdots \\ \vdots & \ddots & \vdots \\ \vdots & \ddots & \vdots \end{bmatrix} \times \begin{bmatrix} \vdots & \ddots & \vdots \\ \vdots & \ddots & \vdots \\ \vdots & \ddots & \vdots \end{bmatrix}$$

$\mathbf{U}: 7 \times 7$
matrix

$\mathbf{S}: 7 \times 9$
diagonal matrix

$\mathbf{V}^T: 9 \times 9$
matrix

- Consider $A_{2,3}$.
- It shows whether User 2 has watched Movie 3.
- How do we compute $A_{2,3}$ using $\mathbf{U}, \mathbf{S}, \mathbf{V}$?
- Let \mathbf{u}_2 be the second row of \mathbf{U} .
- Let \mathbf{v}_3 be the third row of \mathbf{V} .
- Then, $A_{2,3} = \mathbf{u}_2 \mathbf{S}(\mathbf{v}_3)^T =$

$$\sum_{d=1}^7 U_{2,d} S_{d,d} V_{3,d}$$

- Each row \mathbf{u}_i of \mathbf{U} is a vector representation of user i .
- Each row \mathbf{v}_j of \mathbf{V} is a vector representation of movie j .
- $S_{d,d}$ assigns a weight to each dimension d of these vectors.
- Score $A_{i,j}$ is the weighted dot product of \mathbf{u}_i and \mathbf{v}_j , where the weight of each dimension is value $S_{d,d}$.

$$\begin{bmatrix} \vdots & \ddots & \vdots \\ & \ddots & \\ & & \ddots \end{bmatrix} \times \begin{bmatrix} \vdots & \ddots & \vdots \\ & \ddots & \\ & & \ddots \end{bmatrix} \times \begin{bmatrix} \vdots & \ddots & \vdots \\ & \ddots & \\ & & \ddots \end{bmatrix}$$

$\mathbf{U}: 7 \times 7$
matrix

$\mathbf{S}: 7 \times 9$
diagonal matrix

$\mathbf{V}^T: 9 \times 9$
matrix

Modeling Users and Movies

- SVD gives us a way to model users and movies.
 - A user is a vector.
 - A movie is a vector.
 - The score corresponding to a user and a movie is the weighted dot product of these vectors.
- The dimensions are ordered in decreasing order of importance.
 - The first dimension of the user vectors and the movie vectors is the most important for predicting the user/movie score.
 - Each next dimension becomes less important.
- This is similar to PCA projections, where dimensions are also ordered in decreasing order of importance.

Making Predictions

- How can we use SVD to guess the score for a user/movie pair?
- This is similar to predicting missing values using PCA.
 - Using all eigenvectors, backprojection just produces the original input.
 - Using the top few eigenvectors, the backprojection provides a reasonable guess for the hidden pixels.
 - Why? Because the backprojection formula assumes that the missing data follows the statistical patterns of the training data.



Occluded bottom half
(input to PCA).



Backprojection using
10 top eigenvectors.



Backprojection using
all eigenvectors.

Making Predictions

- Suppose that $A_{i,j} = 0$.
 - This means that User i has NOT watched Movie j .
- We would like to use matrices \mathbf{U} , \mathbf{S} , \mathbf{V} to guess if we should recommend Movie j to User i .
- $\mathbf{u}_i \mathbf{S}(\mathbf{v}_j)^T = ???$



Occluded bottom half
(input to PCA).



Backprojection using
10 top eigenvectors.



Backprojection using
all eigenvectors.

Making Predictions

- Suppose that $A_{i,j} = 0$.
 - This means that User i has NOT watched Movie j .
- We would like to use matrices \mathbf{U} , \mathbf{S} , \mathbf{V} to guess if we should recommend Movie j to User i .
- $\mathbf{u}_i \mathbf{S}(\mathbf{v}_j)^T = \mathbf{0}$, since $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$.



Occluded bottom half
(input to PCA).



Backprojection using
10 top eigenvectors.



Backprojection using
all eigenvectors.

Making Predictions

- However, if we use the top few dimensions of \mathbf{u}_i and \mathbf{v}_j , we get something similar to PCA backprojections guessing missing values.
 - These first few dimensions are not enough to perfectly reconstruct matrix \mathbf{A} .
 - Predictions made using these first few dimensions assume that the value we want to predict matches the statistical patterns described by \mathbf{A} , where some users have similar preferences, and some movies have similar ratings.



Occluded bottom half
(input to PCA).



Backprojection using
10 top eigenvectors.



Backprojection using
all eigenvectors.

Full-Dimensional SVD

$$\begin{array}{c}
 \text{Movie 1} \\
 \text{Movie 2} \\
 \text{Movie 3} \\
 \text{Movie 4} \\
 \text{Movie 5} \\
 \text{Movie 6} \\
 \text{Movie 7} \\
 \text{Movie 8} \\
 \text{Movie 9}
 \end{array}
 \begin{array}{c}
 \text{User 1} \\
 \text{User 2} \\
 \text{User 3} \\
 \text{User 4} \\
 \text{User 5} \\
 \text{User 6} \\
 \text{User 7}
 \end{array}
 \begin{bmatrix}
 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0
 \end{bmatrix} =$$

$$\begin{bmatrix}
 \vdots & \ddots & \vdots \\
 & \ddots & \\
 & & \ddots
 \end{bmatrix} \times \begin{bmatrix}
 \vdots & \ddots & \vdots \\
 & \ddots & \\
 & & \ddots
 \end{bmatrix} \times \begin{bmatrix}
 \vdots & \ddots & \vdots \\
 & \ddots & \\
 & & \ddots
 \end{bmatrix}$$

$\mathbf{U}: 7 \times 7$ matrix $\mathbf{S}: 7 \times 9$ diagonal matrix $\mathbf{V}^T: 9 \times 9$ matrix

M - Dimensional SVD

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6	Movie 7	Movie 8	Movie 9	
User 1	0	1	1	1	0	0	1	0	0	=
User 2	1	0	1	0	0	1	0	0	0	
User 3	0	0	0	0	1	0	0	0	1	
User 4	0	1	0	1	0	0	0	1	0	
User 5	1	0	0	0	1	0	0	0	1	
User 6	0	0	1	0	0	0	1	0	1	
User 7	1	0	0	0	0	1	0	0	0	

$$\begin{bmatrix} \vdots & \ddots & \vdots \\ & \ddots & \\ & & \ddots \end{bmatrix} \times \begin{bmatrix} \vdots & \ddots & \vdots \\ & \ddots & \\ & & \ddots \end{bmatrix} \times \begin{bmatrix} \vdots & \ddots & \vdots \\ & \ddots & \\ & & \ddots \end{bmatrix}$$

$U: 7 \times M$

matrix

$S: M \times M$

diagonal matrix

$V^T: M \times 9$

matrix

Testing M -Dimensional SVD

- If this is our ratings matrix A , how can we test SVD to see how well it works?
- We will **hide** some values we know, and see if we can guess them using SVD.
- The values we will hide are marked in red.

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6	Movie 7	Movie 8	Movie 9
User 1	0	1	1	1	0	0	1	0	0
User 2	1	0	1	0	0	1	0	0	0
User 3	0	0	0	0	1	0	0	0	1
User 4	0	1	0	1	0	0	0	1	0
User 5	1	0	0	0	1	0	0	0	1
User 6	0	0	1	0	0	0	1	0	1
User 7	1	0	0	0	0	1	0	0	0

Testing M -Dimensional SVD


- Thus, we get a training matrix data, where the values we hide are set to 0.
- These are movies that we know that the user watched.
- If SVD learned something reasonable, it should predict a high score for those values.

Training Matrix →

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6	Movie 7	Movie 8	Movie 9
User 1	0	1	1	1	0	0	1	0	0
User 2	1	0	1	0	0	1	0	0	0
User 3	0	0	0	0	1	0	0	0	1
User 4	0	1	0	1	0	0	0	1	0
User 5	1	0	0	0	1	0	0	0	0
User 6	0	0	1	0	0	0	0	0	1
User 7	1	0	0	0	0	0	0	0	0

Testing M -Dimensional SVD

- We do SVD on the training matrix.
 - How to do SVD is something we have not discussed yet, we'll get to it.
- We only keep the first 4 columns of U and V .
 - So, $M = 4$.


Training Matrix 

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6	Movie 7	Movie 8	Movie 9
User 1	0	1	1	1	0	0	1	0	0
User 2	1	0	1	0	0	1	0	0	0
User 3	0	0	0	0	1	0	0	0	1
User 4	0	1	0	1	0	0	0	1	0
User 5	1	0	0	0	1	0	0	0	0
User 6	0	0	1	0	0	0	0	0	1
User 7	1	0	0	0	0	0	0	0	0


Training
Matrix

Colors:

- Red: hidden values
- Green: 1 in training matrix.
- Rest: 0 in training matrix



	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6	Movie 7	Movie 8	Movie 9
User 1	0	1	1	1	0	0	1	0	0
User 2	1	0	1	0	0	1	0	0	0
User 3	0	0	0	0	1	0	0	0	1
User 4	0	1	0	1	0	0	0	1	0
User 5	1	0	0	0	1	0	0	0	0
User 6	0	0	1	0	0	0	0	0	1
User 7	1	0	0	0	0	0	0	0	0




0.00	1.00	1.01	1.00	-0.17	0.17	0.63	0.37	0.17
1.12	0.00	1.02	0.00	-0.02	0.69	0.17	-0.17	-0.03
0.06	-0.02	0.08	-0.02	0.97	-0.18	-0.05	0.03	0.99
0.00	1.00	-0.01	1.00	0.18	-0.17	0.37	0.63	-0.17
1.02	0.02	-0.09	0.02	0.86	0.16	-0.12	0.14	0.16
-0.11	0.01	0.94	0.01	0.18	0.16	0.22	-0.21	0.88
0.78	-0.02	0.08	-0.02	0.24	0.26	-0.05	0.03	-0.18

USV^T , with $M = 4$.

Training
Matrix

In USV^T :

- Positions that are 1 in training matrix get high values.
- Positions that are 0 in training matrix get lower values.



	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6	Movie 7	Movie 8	Movie 9
User 1	0	1	1	1	0	0	1	0	0
User 2	1	0	1	0	0	1	0	0	0
User 3	0	0	0	0	1	0	0	0	1
User 4	0	1	0	1	0	0	0	1	0
User 5	1	0	0	0	1	0	0	0	0
User 6	0	0	1	0	0	0	0	0	1
User 7	1	0	0	0	0	0	0	0	0


0.00	1.00	1.01	1.00	-0.17	0.17	0.63	0.37	0.17
1.12	0.00	1.02	0.00	-0.02	0.69	0.17	-0.17	-0.03
0.06	-0.02	0.08	-0.02	0.97	-0.18	-0.05	0.03	0.99
0.00	1.00	-0.01	1.00	0.18	-0.17	0.37	0.63	-0.17
1.02	0.02	-0.09	0.02	0.86	0.16	-0.12	0.14	0.16
-0.11	0.01	0.94	0.01	0.18	0.16	0.22	-0.21	0.88
0.78	-0.02	0.08	-0.02	0.24	0.26	-0.05	0.03	-0.18

USV^T , with $M = 4$.

Training
Matrix

In USV^T :

- Positions of hidden values get relatively high scores.



	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6	Movie 7	Movie 8	Movie 9
User 1	0	1	1	1	0	0	1	0	0
User 2	1	0	1	0	0	1	0	0	0
User 3	0	0	0	0	1	0	0	0	1
User 4	0	1	0	1	0	0	0	1	0
User 5	1	0	0	0	1	0	0	0	0
User 6	0	0	1	0	0	0	0	0	1
User 7	1	0	0	0	0	0	0	0	0


0.00	1.00	1.01	1.00	-0.17	0.17	0.63	0.37	0.17
1.12	0.00	1.02	0.00	-0.02	0.69	0.17	-0.17	-0.03
0.06	-0.02	0.08	-0.02	0.97	-0.18	-0.05	0.03	0.99
0.00	1.00	-0.01	1.00	0.18	-0.17	0.37	0.63	-0.17
1.02	0.02	-0.09	0.02	0.86	0.16	-0.12	0.14	0.16
-0.11	0.01	0.94	0.01	0.18	0.16	0.22	-0.21	0.88
0.78	-0.02	0.08	-0.02	0.24	0.26	-0.05	0.03	-0.18

USV^T , with $M = 4$.

Training
Matrix

In USV^T :

- Consider row 6 col 7.
- The value is 0.22.
- In that row, all other values are lower, except for col 3, where the value was 1 in the training matrix.



	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6	Movie 7	Movie 8	Movie 9
User 1	0	1	1	1	0	0	1	0	0
User 2	1	0	1	0	0	1	0	0	0
User 3	0	0	0	0	1	0	0	0	1
User 4	0	1	0	1	0	0	0	1	0
User 5	1	0	0	0	1	0	0	0	0
User 6	0	0	1	0	0	0	0	0	1
User 7	1	0	0	0	0	0	0	0	0

0.00	1.00	1.01	1.00	-0.17	0.17	0.63	0.37	0.17
1.12	0.00	1.02	0.00	-0.02	0.69	0.17	-0.17	-0.03
0.06	-0.02	0.08	-0.02	0.97	-0.18	-0.05	0.03	0.99
0.00	1.00	-0.01	1.00	0.18	-0.17	0.37	0.63	-0.17
1.02	0.02	-0.09	0.02	0.86	0.16	-0.12	0.14	0.16
-0.11	0.01	0.94	0.01	0.18	0.16	0.22	-0.21	0.88
0.78	-0.02	0.08	-0.02	0.24	0.26	-0.05	0.03	-0.18

USV^T , with $M = 4$.

Training
Matrix

In USV^T :

- So, if we were to recommend a movie for User 6, which one would we recommend?
- The movie with highest score, among all movies that User 6 did not watch.

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6	Movie 7	Movie 8	Movie 9
User 1	0	1	1	1	0	0	1	0	0
User 2	1	0	1	0	0	1	0	0	0
User 3	0	0	0	0	1	0	0	0	1
User 4	0	1	0	1	0	0	0	1	0
User 5	1	0	0	0	1	0	0	0	0
User 6	0	0	1	0	0	0	0	0	1
User 7	1	0	0	0	0	0	0	0	0

0.00	1.00	1.01	1.00	-0.17	0.17	0.63	0.37	0.17
1.12	0.00	1.02	0.00	-0.02	0.69	0.17	-0.17	-0.03
0.06	-0.02	0.08	-0.02	0.97	-0.18	-0.05	0.03	0.99
0.00	1.00	-0.01	1.00	0.18	-0.17	0.37	0.63	-0.17
1.02	0.02	-0.09	0.02	0.86	0.16	-0.12	0.14	0.16
-0.11	0.01	0.94	0.01	0.18	0.16	0.22	-0.21	0.88
0.78	-0.02	0.08	-0.02	0.24	0.26	-0.05	0.03	-0.18


USV^T , with $M = 4$.

Training
Matrix

We would recommend:

- Movies 6 and 9 to User 5.
- Movie 7 to User 6.
- Movie 6 to User 7.

3 out of the 4
recommendations are movies
that these users did watch
(and SVD did not know).



	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6	Movie 7	Movie 8	Movie 9
User 1	0	1	1	1	0	0	1	0	0
User 2	1	0	1	0	0	1	0	0	0
User 3	0	0	0	0	1	0	0	0	1
User 4	0	1	0	1	0	0	0	1	0
User 5	1	0	0	0	1	0	0	0	0
User 6	0	0	1	0	0	0	0	0	1
User 7	1	0	0	0	0	0	0	0	0

0.00	1.00	1.01	1.00	-0.17	0.17	0.63	0.37	0.17
1.12	0.00	1.02	0.00	-0.02	0.69	0.17	-0.17	-0.03
0.06	-0.02	0.08	-0.02	0.97	-0.18	-0.05	0.03	0.99
0.00	1.00	-0.01	1.00	0.18	-0.17	0.37	0.63	-0.17
1.02	0.02	-0.09	0.02	0.86	0.16	-0.12	0.14	0.16
-0.11	0.01	0.94	0.01	0.18	0.16	0.22	-0.21	0.88
0.78	-0.02	0.08	-0.02	0.24	0.26	-0.05	0.03	-0.18

USV^T , with $M = 4$.

Computing SVD

- Let A be any matrix.
- We want to compute the SVD decomposition of A .
- In other words, we want to find matrices U, S, V , such that:
 - $A = USV^T$
 - U is an $m \times m$ square matrix.
 - S is an $m \times n$ diagonal matrix (only entries $S_{i,i}$ are non-zero).
 - $S_{i+1,i+1} \leq S_{i,i}$. So, the diagonal entries of S appear in decreasing order.
 - V is an $n \times n$ matrix.

Computing SVD

- Let A be any matrix.
- We want to compute the SVD decomposition of A .
 - $A = USV^T$
- Then:
 - The columns of U are the eigenvectors of AA^T , in decreasing order of the corresponding eigenvalues.
 - The columns of V are the eigenvectors of $A^T A$, in decreasing order of the corresponding eigenvalues.
 - The diagonal entries of S are the square roots of the eigenvalues of AA^T , in decreasing order.
- You can compute U, S, V via the power method.
- Better methods are also available.

The MovieLens Dataset

- The Netflix dataset is not publicly available anymore.
- The **MovieLens** dataset is a similar dataset that is publicly available.
- It has:
 - 20 million movie ratings.
 - 138,000 users.
 - 27,000 movies.
- There is a "small" version of that dataset, with:
 - 100,000 movie ratings.
 - 700 users.
 - 9,000 movies.

Trying SVD on MovieLens

- We will use the "small" MovieLens dataset to repeat the experiment we did before with our toy example.
- We convert the ratings matrix to a binary (0/1) matrix.
 - 0 stands for "user did not watch movie".
 - 1 stands for "user watched movie".
 - 98.4% of the binary matrix values are equal to 0.
 - 1.6% of the binary matrix values are equal to 1.
- We want to use SVD to recommend movies to users.
- How can we test how well SVD works?

Trying SVD on MovieLens

- We create a training matrix, by randomly choosing half of the 1 values, and "hiding" them (setting them to 0).
- These hidden values are 0 in the training data, but we know that users actually watched those movies.
 - We can assume that users were interested in those movies, and it would have been a good idea to recommend those movies to those users.
- If SVD gives high scores to those movies, it means that it has learned a good model for making movie recommendations.

Issues with Evaluating Accuracy

- In general, evaluating prediction accuracy is a **non-trivial task**.
 - This is an issue in general classification tasks as well.
 - This is simply our first encounter with some of the complexities that arise.
- Oftentimes people do not understand the underlying issues, and report numbers that are impressive, but meaningless.

Issues with Evaluating Accuracy

- For example: consider the Movielens.
- Suppose we build a model that says that each user is not interested in any movie that is not marked as "watched" in the training data.
- How accurate is that model?
 - How do we evaluate that accuracy?
- Intuitively, we know that such a model is pretty useless.
- We know for a fact that some zeros in the training matrix were corresponding to "hidden" ones.
- How can we capture quantitatively the fact that the model is very bad?

Issues with Evaluating Accuracy

0.00	1.00	1.01	1.00	− 0.17	0.17	0.63	0.37	0.17
1.12	0.00	1.02	0.00	− 0.02	0.69	0.17	− 0.17	− 0.03
0.06	− 0.02	0.08	− 0.02	0.97	− 0.18	− 0.05	0.03	0.99
0.00	1.00	− 0.01	1.00	0.18	− 0.17	0.37	0.63	− 0.17
1.02	0.02	− 0.09	0.02	0.86	0.16	− 0.12	0.14	0.16
− 0.11	0.01	0.94	0.01	0.18	0.16	0.22	− 0.21	0.88
0.78	− 0.02	0.08	− 0.02	0.24	0.26	− 0.05	0.03	− 0.18

- Our training matrix contains some ones and some zeros.
 - The ones are known values.
 - The zeros may be true zeros, or they may be true ones that were "hidden".
- If we use M -dimensional SVD, we get an estimated score matrix.
- Remember, for our toy example, we got the matrix shown above.
- How can we convert those values to predictions?

Issues with Evaluating Accuracy

- We can choose a threshold T .
- All values $\geq T$ correspond to prediction "1" (i.e., to prediction that we should recommend that movie).
- All values $< T$ correspond to prediction "0" (i.e., to prediction that we should not recommend that movie).
- How should we choose a threshold? Based on accuracy? We end up with the same issue: how should we measure accuracy?

Issues with Evaluating Accuracy

- Going back to the MovieLens dataset:
- Once we get predictions of ones and zeros, we can compare them against the ones and zeros of the true data matrix.
 - In the true data matrix the "hidden" ones show as actual ones.
- In measuring accuracy, we will exclude the positions where the training matrix contains "1" values.
 - The goal is to use the predictions to recommend movies.
 - If we have a "1" in the training matrix, we know that the user watched that movie, so there is no point in recommending that movie to that user.

Issues with Evaluating Accuracy

- So, suppose that we choose a threshold of $+\infty$.
- As humans, we know that this is a bad idea.
 - All predictions are "0", we do not make any recommendations.
- What is the accuracy?
- The true score matrix has about 6 million values.
 - 50,000 of those values are set to "1" in both the true score matrix and the training matrix.
 - We do not use those values to evaluate accuracy.
 - About 6 million values are set to "0" in the training matrix.
 - These are the values that we will use to evaluate accuracy.

Issues with Evaluating Accuracy

- So, we have about 6,000,000 predictions to evaluate.
- All those predictions are equal to 0.
 - Because we chose a threshold of $+\infty$.
- Based on the true score matrix:
 - 50,000 of those predictions are wrong, because they correspond to hidden ones.
 - About 6,000,000 predictions are right.
- Classic mistake #1: evaluate accuracy using this formula:
 - $$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Number of total predictions}} \cong 99.17\%$$
- 99.17% sounds pretty impressive, but it is misleading.
 - The system is useless, it does not make any recommendations.

Issues with Evaluating Accuracy

- One issue is that the classes are **asymmetrically distributed**.
- There are way more "0" values than "1" values.
- Many machine learning problems are like that.
- Another example: face detection in images.
 - In many datasets, more than 99.9% of image windows are NOT faces.
 - A face detector that never detects any faces is 99.9% accurate.
- Another issue is that the final prediction depends on an arbitrary threshold.

Balanced Accuracy Rate

- When we have asymmetric class distributions, a useful measure is the **balanced accuracy rate**:
 - First, calculate the accuracy rate separately for objects belonging to each class.
 - The balanced accuracy rate is the average of those class-specific error rates.
- In our example, using a threshold of $+\infty$:
 - The overall accuracy is 99.17%.
 - This is an impressive and useless number.
 - The balanced accuracy rate is 50%.
 - 100% for true 0 values, and 0% for true 1 values.
 - This is a far more informative measure of accuracy.

Precision vs. Recall

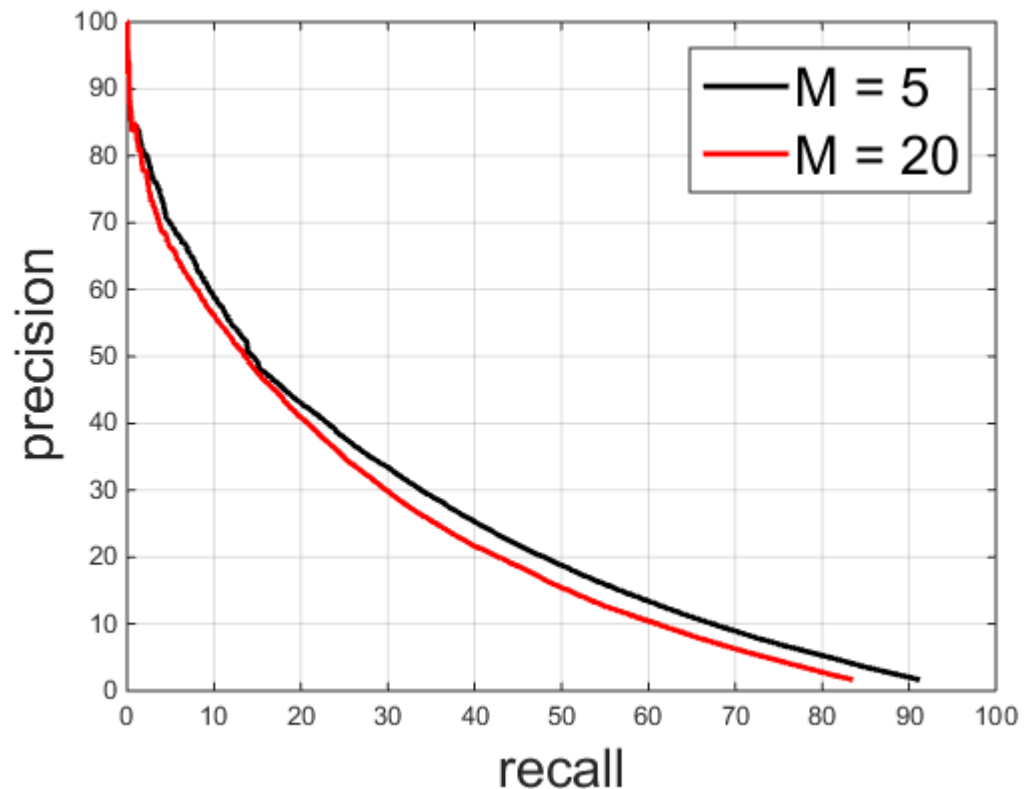
- Movie recommendation can be thought of as a **retrieval problem**.
 - There are lots of possible items to recommend.
 - Only few of those items are relevant.
- Any system that makes recommendations can be evaluated using two measures:
 - **Precision**: the fraction of retrieved items that are relevant.
 - In our case: $\frac{\text{number of estimated ones that are true ones}}{\text{number of estimated ones}}$.
 - **Recall**: the fraction of relevant items that are retrieved.
 - In our case: $\frac{\text{number of estimated ones that are true ones}}{\text{number of true ones}}$.

Precision vs. Recall

- Suppose that we have two methods, A and B.
- Suppose that each of these methods uses a threshold to produce the final results.
- Classic mistake #2:
 - "Method A achieved a precision of 90% and a recall of 80%".
 - "Method B achieved a precision of 92% and a recall of 60%".
- If the results of A and B depend on thresholds, **you should never report results using a single threshold.**
- Instead, you should show a **precision vs. recall plot.**

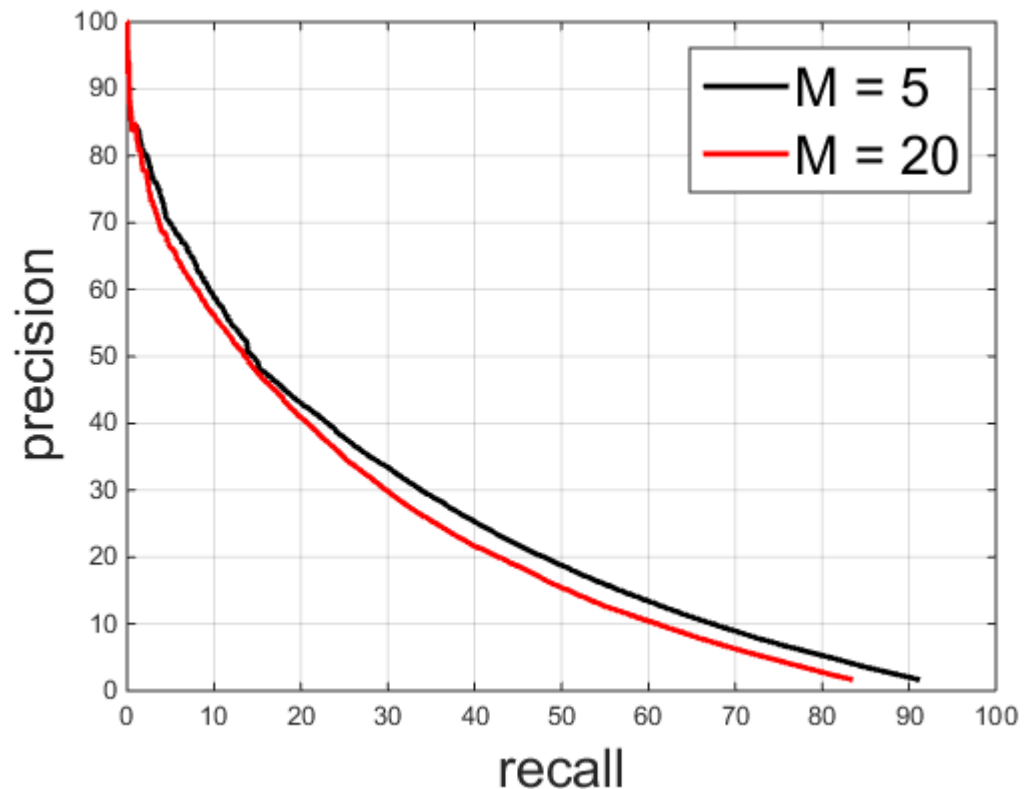
Precision vs. Recall

- Here we see an example figure of precision vs. recall.
 - It shows, for many different thresholds, the precision and recall rates attained with SVD on the MovieLens dataset.
- We have one plot for $M = 5$, and one for $M = 20$.
- Does this plot tell us which choice of M is better?
 - Yes! $M = 5$ gives **consistently better results** than $M = 20$.



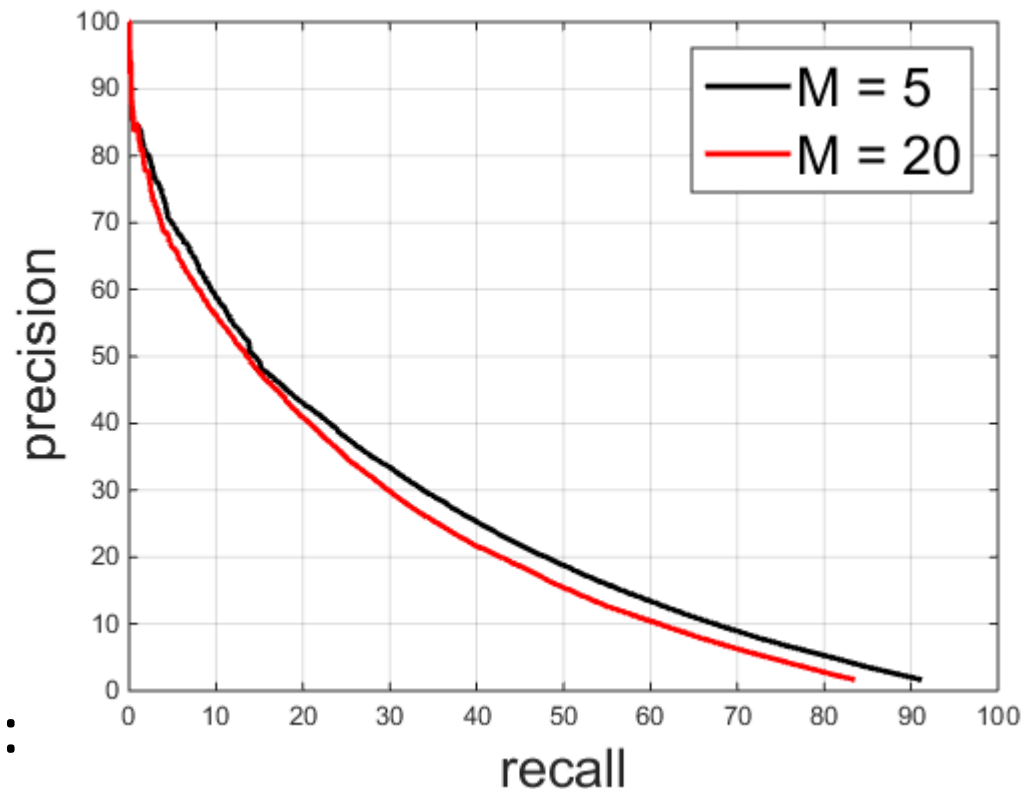
Choosing a Threshold

- The choice of threshold makes a big difference.
- A low threshold leads to:
 - A higher number of total recommendations.
 - Consequently, higher recall (higher number of correct items are recommended).
 - A smaller percentage of correct recommendations among the recommendations that are made.
 - Consequently, lower precision.



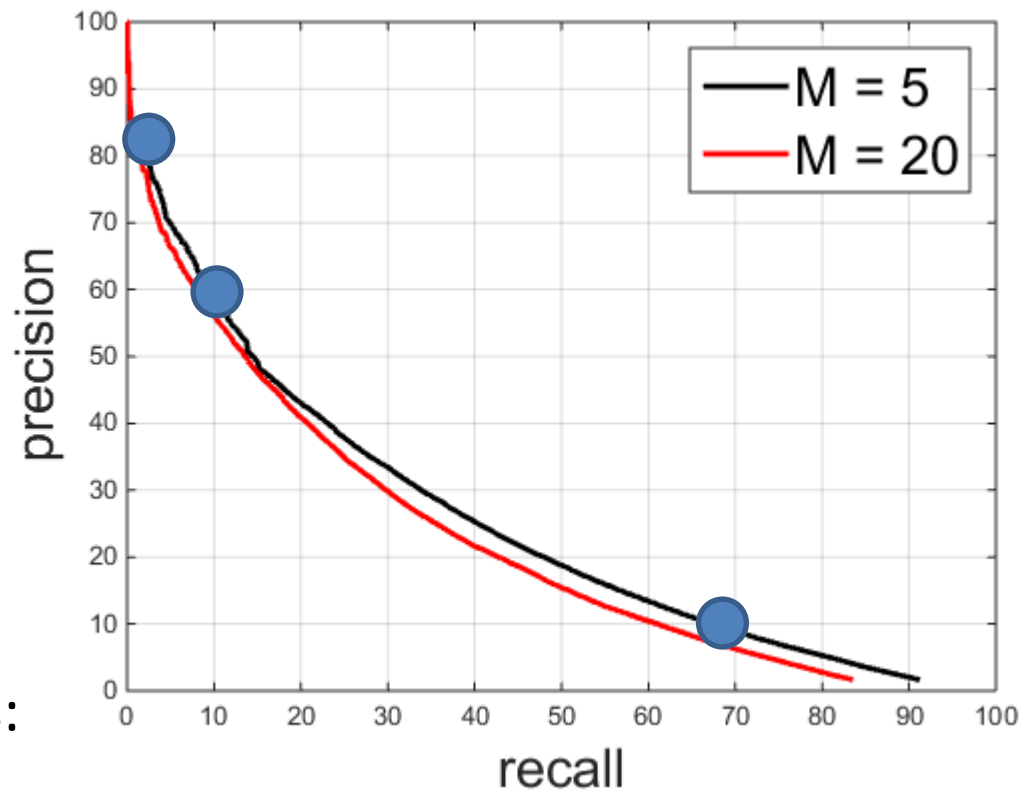
Choosing a Threshold

- The choice of threshold makes a big difference.
- A high threshold leads to:
 - A lower number of total recommendations.
 - Consequently, lower recall (lower number of correct items are recommended).
 - A larger percentage of correct recommendations among the recommendations that are made.
 - Consequently, higher precision.



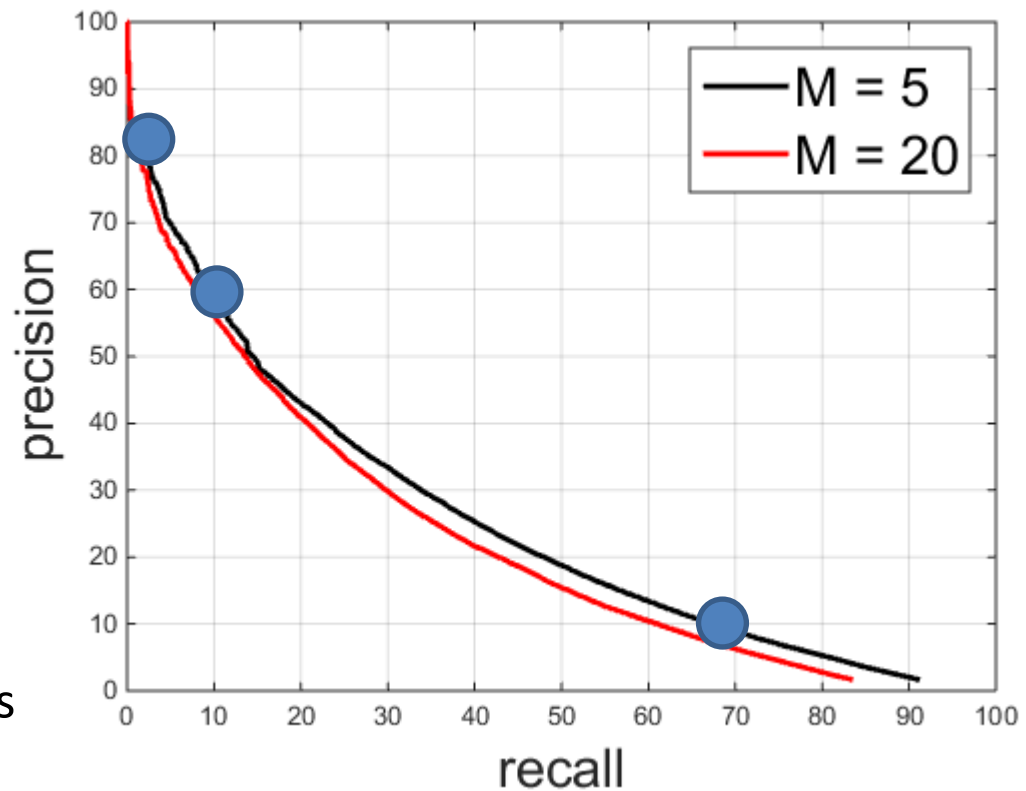
Choosing a Threshold

- Each point on the curve represents a trade-off between precision and recall.
- For example, one point gives:
 - precision 10.1%.
 - recall 67.3%.
- Another point corresponds to:
 - precision 59.0%.
 - recall 9.9%.
- Another point corresponds to:
 - precision 84.7%.
 - recall 1.1%.



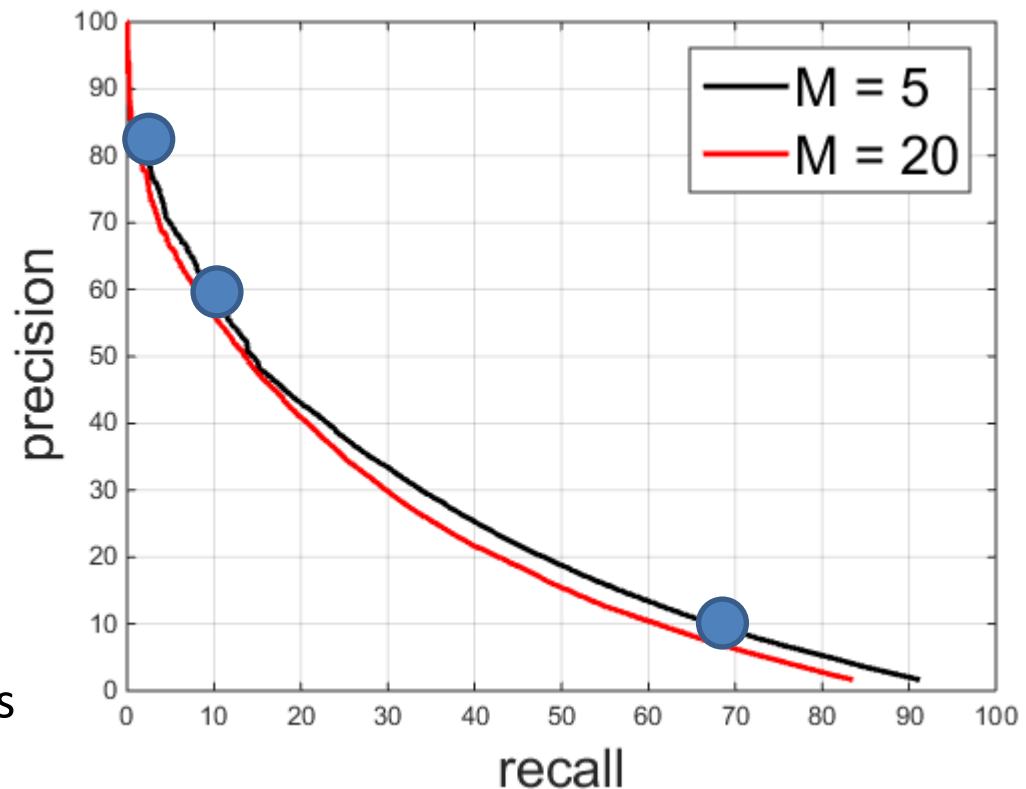
Choosing a Threshold

- Let's look at this point:
 - precision 10.1%.
 - recall 67.3%.
- Recall 67.3% means:
 - Of all correct recommendations (which are 50,002), the system identifies and provides 67.3% of them to the user.
 - So, the system makes 33651 correct recommendations (50.15 per user).
- Precision 10.1% means:
 - Those 50.15 correct recommendations per user are 10.1% of the total recommendations made per user.
 - The number of total recommendations made is $50.15 * \frac{100}{10.1} = 496.53$.
- The average user gets 497 suggestions, 50 of those are correct.



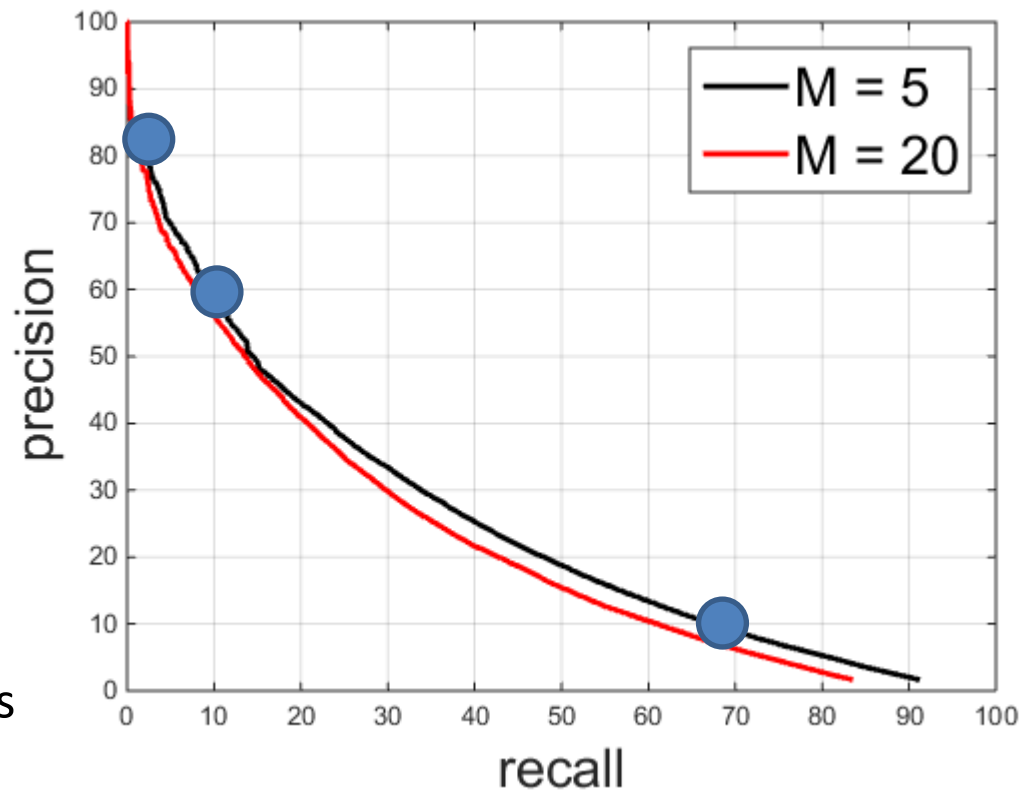
Choosing a Threshold

- Let's look at this point:
 - precision 59.0%.
 - recall 9.9%.
- Recall 9.9% means:
 - Of all correct recommendations (which are 50,002), the system identifies and provides 9.9% of them to the user.
 - So, the system makes 4950 correct recommendations (7.38 per user).
- Precision 59.0% means:
 - Those 7.38 correct recommendations per user are 59.0% of the total recommendations made per user.
 - The number of total recommendations made is $7.38 * \frac{100}{59} = 12.51$.
- The average user gets 12.5 suggestions, 7.4 of those are correct.⁵³

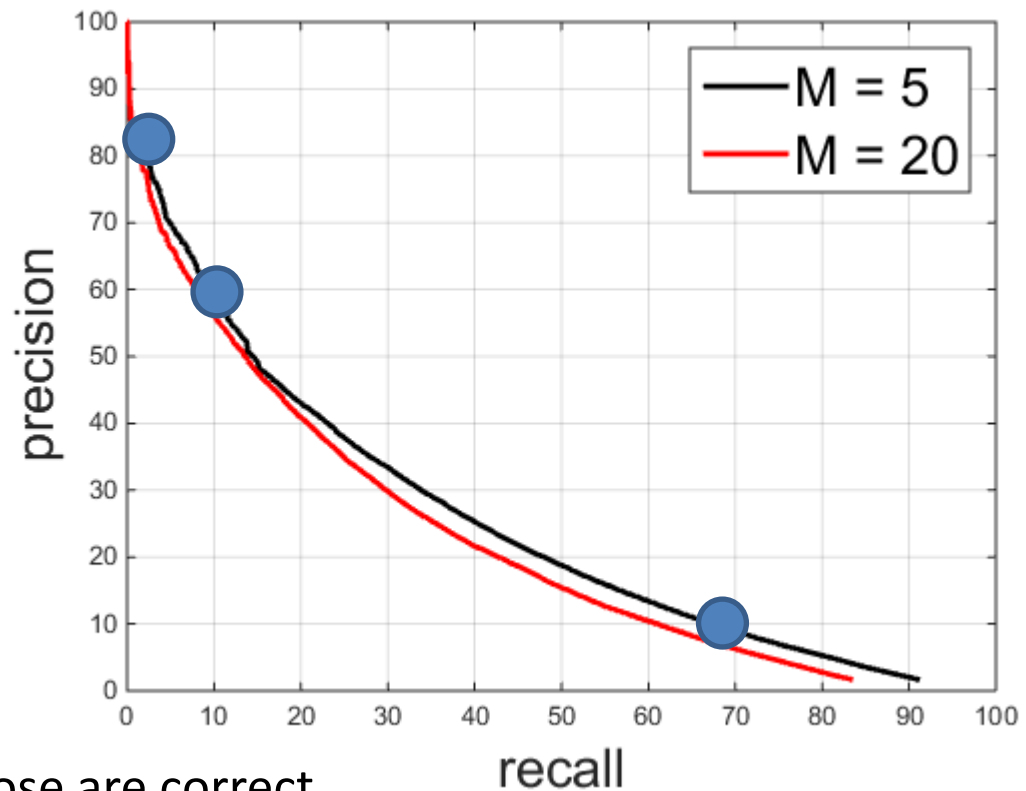


Choosing a Threshold

- Let's look at this point:
 - precision 84.7%.
 - recall 1.1%.
- Recall 1.1% means:
 - Of all correct recommendations (which are 50,002), the system identifies and provides 1.1% of them to the user.
 - So, the system makes 550 correct recommendations (0.82 per user).
- Precision 84.7% means:
 - Those 0.82 correct recommendations per user are 84.7% of the total recommendations made per user.
 - The number of total recommendations made is $0.82 * \frac{100}{84.7} = 0.97$.
- The average user gets 0.97 suggestions, 0.82 of those are correct.



Choosing a Threshold



- So, we have many choices, some of which are:
 - 497 suggestions/user, 50 of those are correct.
 - 12.5 suggestions/user, 7.4 of those are correct.
 - 0.97 suggestions/user, 0.82 of those are correct.
- Which of those choices would you prefer as a user?
 - I would personally prefer the middle option, 7.4 correct suggestions out of 12.5 suggestions.
 - 0.97 suggestions per user is too small.
 - 50 correct recommendations out of 497 is also too small.

The Sparsity of Ratings

- In many cases, the vast majority of the $A_{U,M}$ values are missing.
- For example, in the training set for the Netflix Prize:
 - There were 480,189 users.
 - There were 17,770 movies.
 - The average user in the dataset had rated somewhat over 200 movies (about 1.1% of all movies).
 - The average movie was rated by somewhat over 5,000 users (about 1.1% of all users).
 - Therefore, on the Netflix dataset, about 98.9% of the $A_{U,M}$ values were missing.

The Sparsity of Ratings

- As another example, Amazon has millions of users, and sells tens to hundreds of millions of products.
- It is hard to imagine that any user has bought more than a very small fraction of all available products.
- So, for the rest of the products, the system does not really know if the user is interested in them or not.

Dealing with Sparse Data

- In our MovieLens example, we represented missing data with 0 values.
 - Our training matrix had 0 values for all (user, movie) pairs where the user did not watch that movie.
- However, our goal is not to figure out which movies the user *has watched*.
 - Our goal is to figure out which movies the user may *want to watch*.
- Therefore, instead of representing not-watched movies with 0 values, we can treat those values as **missing data**.

SVD with Missing Data

- SVD requires a matrix A where all values are specified.
- If any value is missing, we cannot do SVD.
 - Even worse, if 99% of the values are missing.
- However, there are workarounds.
- We will look at one such method, proposed in this paper:

Dohyun Kim and Bong-Jin Yum. 2005. Collaborative filtering based on iterative principal component analysis. *Expert Systems with Applications*, 28, 4 (May 2005), 823-830.

Using Averages for Missing Values

- Suppose that matrix A contains user ratings of movies.
- Value $A_{U,M}$, if not missing, is the rating that user U gave to movie M .
- Suppose that a certain value $A_{U,M}$ is missing.
- This value can be filled in with a reasonable guess, computed as follows:
 - Let S_1 be the average of all ratings that user U has provided.
 - Let S_2 be the average of all ratings that movie M has received.
 - Our guess for $A_{U,M}$ is the average of S_1 and S_2 .

Iterative SVD

- Once we fill in all missing values as described, we can do SVD, to obtain M -dimensional representations of users and movies.
- Using these M -dimensional representations, we can (hopefully) provide a better guess for the original missing values.
- How?

Iterative SVD

- Once we fill in all missing values as described, we can do SVD, to obtain M -dimensional representations of users and movies.
- Using these M -dimensional representations, we can (hopefully) provide a better guess for the original missing values.
 - Remember, using SVD, each rating is computed as a weighted dot product between the user vector and the movie vector.
- So, using these dot products, we update our estimate for our missing values.
- Then, we can do SVD again to get even better estimates.

SVD Recap

- SVD is a mathematical method that factorizes a matrix into a product of three matrices with specific constraints.
- This factorization is useful in recommendation systems:
 - One matrix represents users.
 - One matrix represents products.
 - User/product scores are represented as weighted dot products between user vectors and product vectors.
- The training data usually contains lots of missing values.
- To use SVD, we need to first fill in those values somehow (for example, by looking at average scores given by a user, and average scores received by a product).
- Iterative SVD allows us to update our estimate of the missing values, using previous invocations of SVD.

Recommendation Systems

- We have looked at a very simple solution for recommendation systems, using SVD.
- Much more sophisticated (and accurate) methods are available.
- For more information, you can search on the Web for things like:
 - Netflix prize.
 - Recommendation systems.