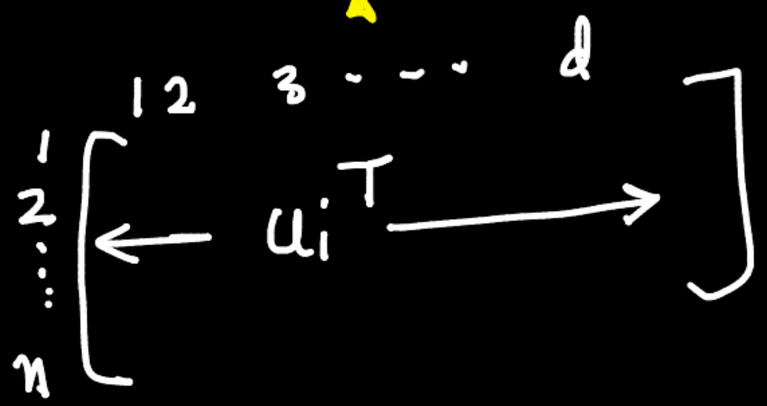
Topics:

1. Recap of MF for RecSys
2. NMF ✓
3. Netflix Prize solution → Paper
4. Code: RecSys ✓
5. MF: feature engineering
   - Text ✓
   - Images ✓
   - Entities → user, webpage, product, video ...
6. Market-basket Analysis (ARM, Apriori algo) ✓

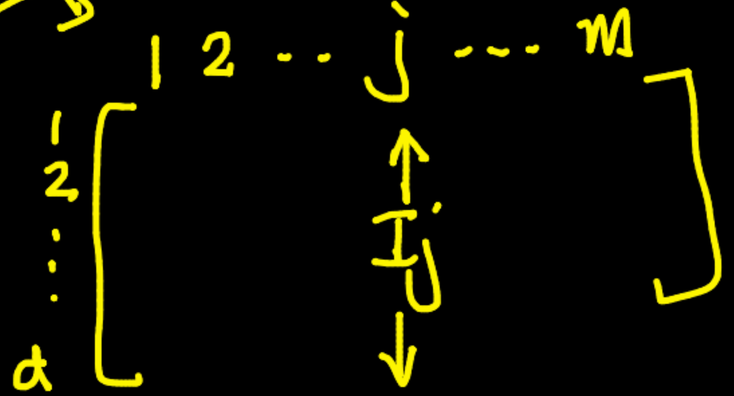
Recap

Given

$A_{n \times m}$ = Found $B_{n \times d} \cdot C_{d \times m}$ ← 2 matrices



$B_i = u_i \in \mathbb{R}^d$



$I_j \in \mathbb{R}^d = g_j$

Opt:

$$\min_{B_i, G_j} \sum_{i,j} (A_{ij} - B_i^T G_j)^2$$

$A_{ij} \neq NULL$

- SGD
- Coordinate descent

d → hyper-params

- elbow method
- train-test

Q

dim red

PCA: $S_{d\times d} = W \Lambda W^T$ ← $X^T X$ ← $X_{n\times d}$ → eig-vec of S

≈

SVD: $X_{n\times d} = U_{n\times n} \Sigma_{n\times d} V^T_{d\times d}$

Σ → diag

U → eigvec of $(X X^T)_{n\times n}$

V^T → eigvec of $(X^T X)_{d\times d} = S_{d\times d}$

$\begin{bmatrix} s_1 & & & \\ & \ddots & & \\ & & s_d & \\ & & & \ddots \end{bmatrix}$

Hard/soft

K-means is a special type of MF

$X \approx Z C$

$X_{n \times d}$, $Z_{n \times K}$, $C_{K \times d}$

Z: cluster centroids

Hard

$$z_{ij} = \begin{cases} 1 & \text{if } x_i \in S_j \\ 0 & \text{o/w} \end{cases}$$

$\sum_j z_{ij} = 1 \quad \forall i$

Cluster assignment: Matrix

Constraints

$Z = [1, 2, \ldots, j, \ldots, K]$

$n \times K$

[Non-negative MF]

$A_{n \times m} = B_{n \times d} \cdot C_{d \times m}$

Bij] elements of A & C

Cij

↳ non-negative ≥ 0

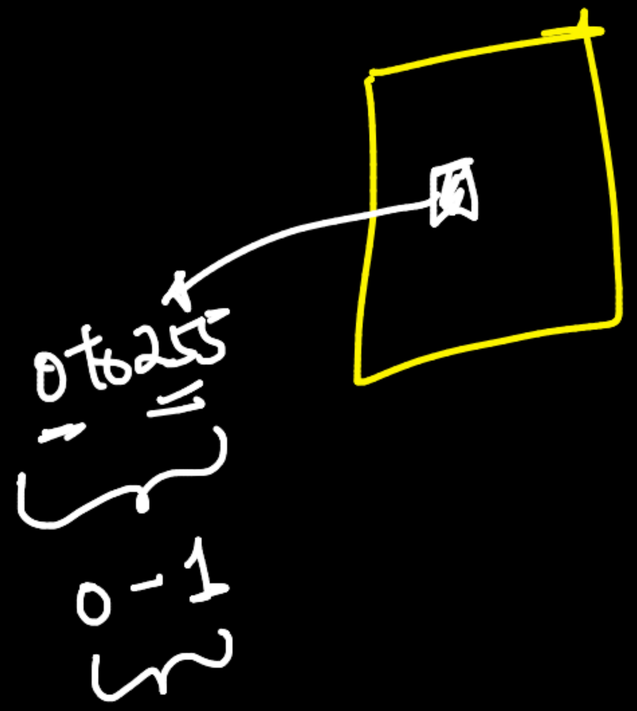
[why?] → NMF: generalization of clustering

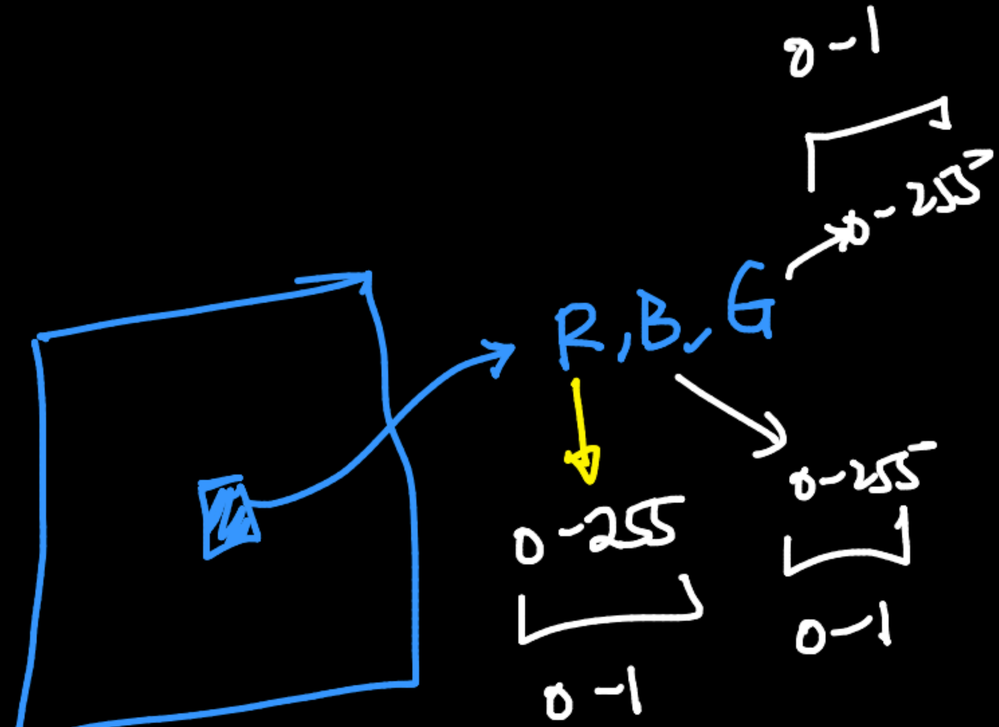
→ Images → visualize

↳ interpretability

MF ⊇ NMF ⊇ K-means

hard/soft



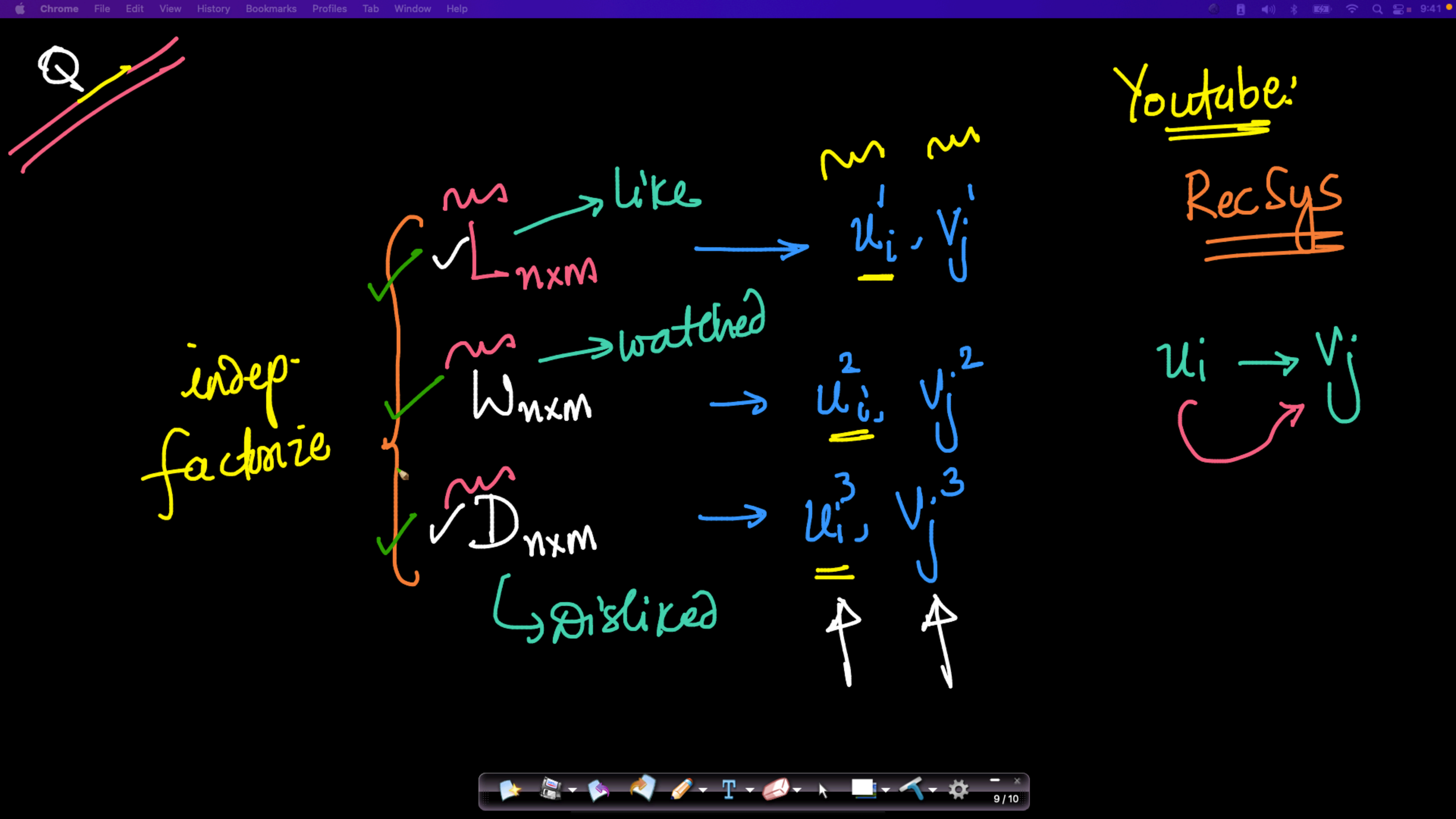


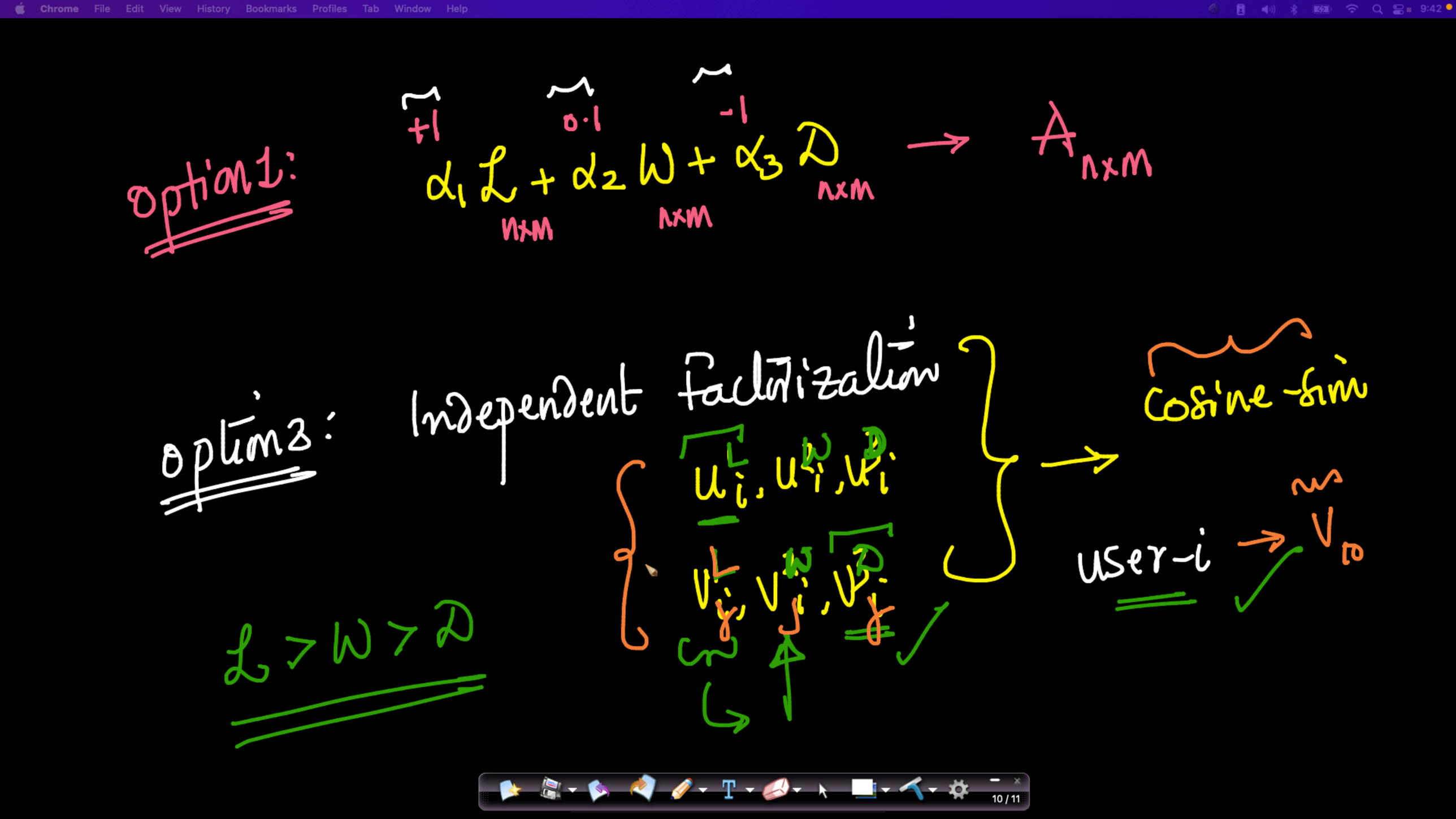
Eigen-faces ← MF: feature engineering

$$\min_{B,C} \|A - B \cdot C\|_F^2 \equiv \sum_{ij} (A_{ij} - B_i^T C_j)^2$$

s.t $B_{ij} \geq 0 \quad \forall i,j$

$C_{ij} \geq 0 \quad \forall i,j$ ✓


Youtube:
RecSys
indep-
factorize
like
watched
disliked
$L_{n \times m}$
$W_{n \times m}$
$D_{n \times m}$
u_i^1, v_j^1
u_i^2, v_j^2
u_i^3, v_j^3
$u_i \rightarrow v_j$


option1:
+1
0.1
-1
$\alpha_1 \mathcal{L} + \alpha_2 W + \alpha_3 D \rightarrow A_{n \times m}$
$n \times m$
option2:
Independent factorization
$u_i^{\mathcal{L}}, u_i^{W}, u_i^{D}$
$v_j^{\mathcal{L}}, v_j^{W}, v_j^{D}$
Cosine-sim
user-i
$\mathcal{L} > W > D$
10 / 11

$$U_{100}^{T} \cdot V_{10} = \mathcal{L}_{100,10}$$

$u_i^L \in \mathbb{R}^d$

$u_i^W \in \mathbb{R}^d$

→ Clustering
→ t-SNE/UMAP } ✓

$\mathcal{L}_{n \times m} \rightarrow 0$ or 1 ✓ ✓

Metric : RMSE

u, m → 0 to 5

Netflix Prize

2008-09

$1MM

never productionized

final-solution → tons of smaller-models + bagging/boosting

10% improvement on RMSE

Paper's Notation

p_u ← u_i : user-i → n-users

q_i ← m_j : mover-j → m-items

r_{ij} : rating

{ $\mathcal{D}_{TR}$

$\mathcal{D}_{Te}$

$u_i, m_j \rightarrow ?$

Simplest:

$$\min_{q_i, p_u} \sum_{u,i} (r_{ui} - q_i^T p_u)^2$$

$p_u \in \mathbb{R}^d$

$q_i \in \mathbb{R}^d$

d - dim

$$\min_{q_i, p_u} \sum_{u,i} (r_{ui} - q_i^T p_u)^2 + \lambda \left(\sum_i \|q_i\|^2 + \sum_u \|p_u\|^2 \right)$$

example, Netflix collects star ratings for movies, and TiVo users indicate their preferences for TV shows by pressing thumbs-up and thumbs-down buttons. We refer to explicit user feedback as *ratings*. Usually, explicit feedback comprises a sparse matrix, since any single user is likely to have rated only a small percentage of possible items.

One strength of matrix factorization is that it allows incorporation of additional information. When explicit feedback is not available, recommender systems can infer user preferences using *implicit feedback*, which indirectly reflects opinion by observing user behavior including purchase history, browsing history, search patterns, or even mouse movements. Implicit feedback usually denotes the presence or absence of an event, so it is typically represented by a densely filled matrix.

A BASIC MATRIX FACTORIZATION MODEL

Matrix factorization models map both users and items to a joint latent factor space of dimensionality f, such that user-item interactions are modeled as inner products in that space. Accordingly, each item i is associated with a

Earlier systems relied on imputation to fill in missing ratings and make the rating matrix dense.[2] However, imputation can be very expensive as it significantly increases the amount of data. In addition, inaccurate imputation might distort the data considerably. Hence, more recent works[3-6] suggested modeling directly the observed ratings only, while avoiding overfitting through a regularized model. To learn the factor vectors (p_u and q_i), the system minimizes the regularized squared error on the set of known ratings:

$$\min_{q^*,p^*} \sum_{(u,i)\in\kappa} (r_{ui} - q_i^T p_u)^2 + \lambda(\| q_i \|^2 + \| p_u \|^2) \qquad (2)$$

Here, κ is the set of the (u,i) pairs for which r_{ui} is known (the training set).

The system learns the model by fitting the previously observed ratings. However, the goal is to generalize those previous ratings in a way that predicts future, unknown ratings. Thus, the system should avoid overfitting the observed data by regularizing the learned parameters, whose magnitudes are penalized. The constant λ controls

associated prediction error

$$e_{ui} \overset{def}{=} r_{ui} - q_i^T p_u.$$

Then it modifies the parameters by a magnitude proportional to γ in the opposite direction of the gradient, yielding:

- $q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i)$
- $p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u)$

reg-Coeff

learning rate

This popular approach[4-6] combines implementation ease with a relatively fast running time. Yet, in some cases, it is beneficial to use ALS optimization.

Alternating least squares

Because both q_i and p_u are unknowns, Equation 2 is not convex. However, if we fix one of the unknowns, the optimization problem becomes quadratic and can be solved optimally. Thus, ALS techniques rotate between fixing the q_i's and fixing the p_u's. When all p_u's are fixed, the system recomputes the q_i's by solving a least-squares problem, and vice versa. This ensures that each st

item biases can explain, subjecting only the true interaction portion of the data to factor modeling. A first-order approximation of the bias involved in rating r_{ui} is as follows:

$$b_{ui} = \mu + b_i + b_u \quad (3)$$

The bias involved in rating r_{ui} is denoted by b_{ui} and accounts for the user and item effects. The overall average rating is denoted by μ; the parameters b_u and b_i indicate the observed deviations of user u and item i, respectively, from the average. For example, suppose that you want a first-order estimate for user Joe's rating of the movie *Titanic*. Now, say that the average rating over all movies, μ, is 3.7 stars. Furthermore, *Titanic* is better than an average movie, so it tends to be rated 0.5 stars above the average. On the other hand, Joe is a critical user, who tends to rate 0.3 stars lower than the average. Thus, the estimate for *Titanic*'s rating by Joe would be 3.9 stars (3.7 + 0.5 − 0.3). Biases extend Equation 1 as follows:

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u \quad (4)$$

is broken down into its four

have rated only a small percentage of possible items.

One strength of matrix factorization is that it allows incorporation of additional information. When explicit feedback is not available, recommender systems can infer user preferences using *implicit feedback*, which indirectly reflects opinion by observing user behavior including purchase history, browsing history, search patterns, or even mouse movements. Implicit feedback usually denotes the presence or absence of an event, so it is typically represented by a densely filled matrix.

A BASIC MATRIX FACTORIZATION MODEL

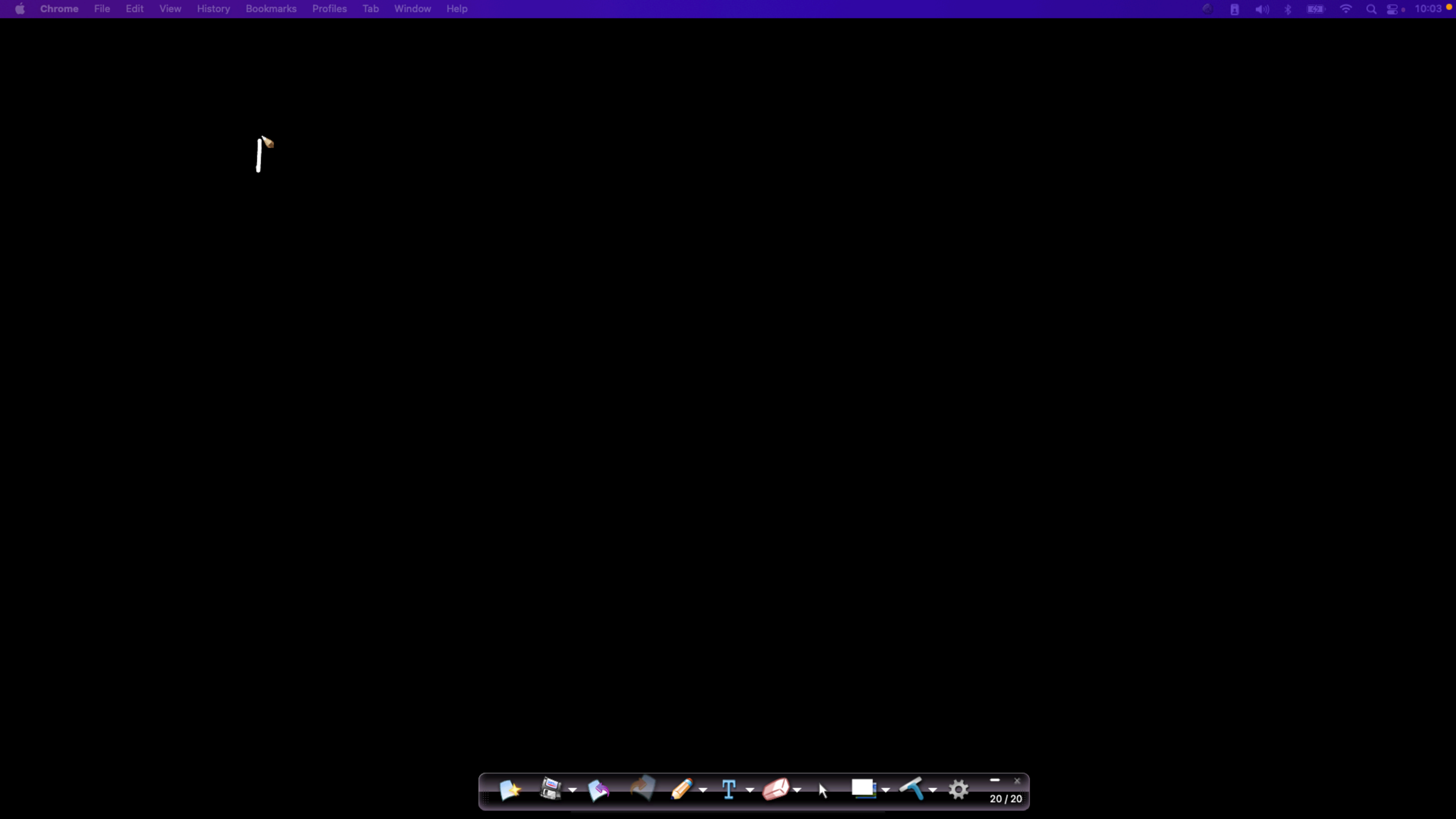
Matrix factorization models map both users and items to a joint latent factor space of dimensionality f, such that user-item interactions are modeled as inner products in that space. Accordingly, each item i is associated with a

works[5,6] suggested modeling directly the observed ratings only, while avoiding overfitting through a regularized model. To learn the factor vectors (p_u and q_i), the system minimizes the regularized squared error on the set of known ratings:

$$\min_{q^*,p^*} \sum_{(u,i)\in\kappa} (r_{ui} - q_i^T p_u)^2 + \lambda(\| q_i \|^2 + \| p_u \|^2) \quad (2)$$

Here, κ is the set of the (u,i) pairs for which r_{ui} is known (the training set).

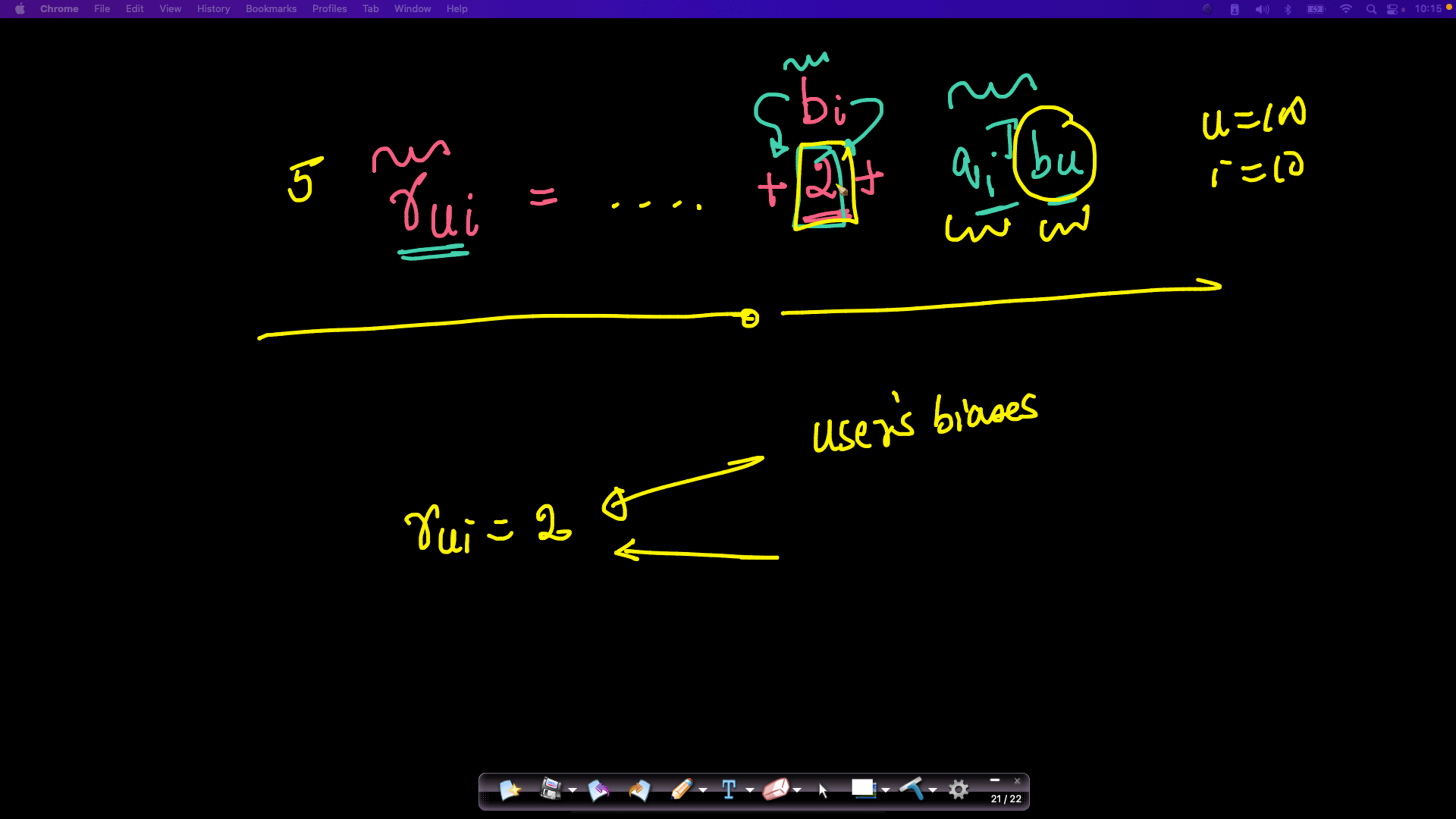
The system learns the model by fitting the previously observed ratings. However, the goal is to generalize those previous ratings in a way that predicts future, unknown ratings. Thus, the system should avoid overfitting the observed data by regularizing the learned parameters, whose magnitudes are penalized. The constant λ controls


Chrome File Edit View History Bookmarks Profiles Tab Window Help
10:03
20 / 20

Intuition

1. avg rating in the Netflix eco-system → μ
2. avg-rating per movie → b_i
3. avg rating given by the user → b_u

$$r_{ui} = \mu + b_u + b_i + q_i^T p_u$$


5
r_{ui} = ...
b_i
+ 21 +
q_i T
b_u
u=10
i=10
user's biases
$r_{ui} = 2$

Diy

$$\min_{p_u, q_i, \mu, b_u, b_i} \sum_{u,i} \left(r_{ui} - \mu - b_u - b_i - p_u^T q_i \right)^2 + \lambda \sum \left\{ \|q_i\|^2 + \|b_u\|^2 + \mu^2 + b_u^2 + b_i^2 \right\}$$

scalars

params $\{ p_u, q_i, \mu, b_u, b_i \}$

w_{ui} → more recent ; more weightage

optimally. Thus, ALS techniques rotate between fixing the q_i's and fixing the p_u's. When all p_u's are fixed, the system recomputes the q_i's by solving a least-squares problem, and vice versa. This ensures that each step decreases Equation 2 until convergence.[8]

While in general stochastic gradient descent is easier and faster than ALS, ALS is favorable in at least two cases. The first is when the system can use parallelization. In ALS, the system computes each q_i independently of the other item factors and computes each p_u independently of the other user factors. This gives rise to potentially massive parallelization of the algorithm.[9] The second case is for systems centered on implicit data. Because the training set cannot be considered sparse, looping over each single training case—as gradient descent does—would not be practical. ALS can efficiently handle such cases.[10]

ADDING BIASES

One benefit of the matrix factorization approach to collaborative filtering is its flexibility in dealing with various

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u \quad (4)$$

Here, the observed rating is broken down into its four components: global average, item bias, user bias, and user-item interaction. This allows each component to explain only the part of a signal relevant to it. The system learns by minimizing the squared error function:[4,5]

$$\min_{p^*,q^*,b^*} \sum_{(u,i)\in\kappa} (r_{ui} - \mu - b_u - b_i - p_u^T q_i)^2 + \lambda$$
$$(\| p_u \|^2 + \| q_i \|^2 + b_u^2 + b_i^2) \quad (5)$$

Since biases tend to capture much of the observed signal, their accurate modeling is vital. Hence, other works offer more elaborate bias models.[11]

ADDITIONAL INPUT SOURCES

Often a system must deal with the cold start problem, wherein many users supply very few ratings, making it

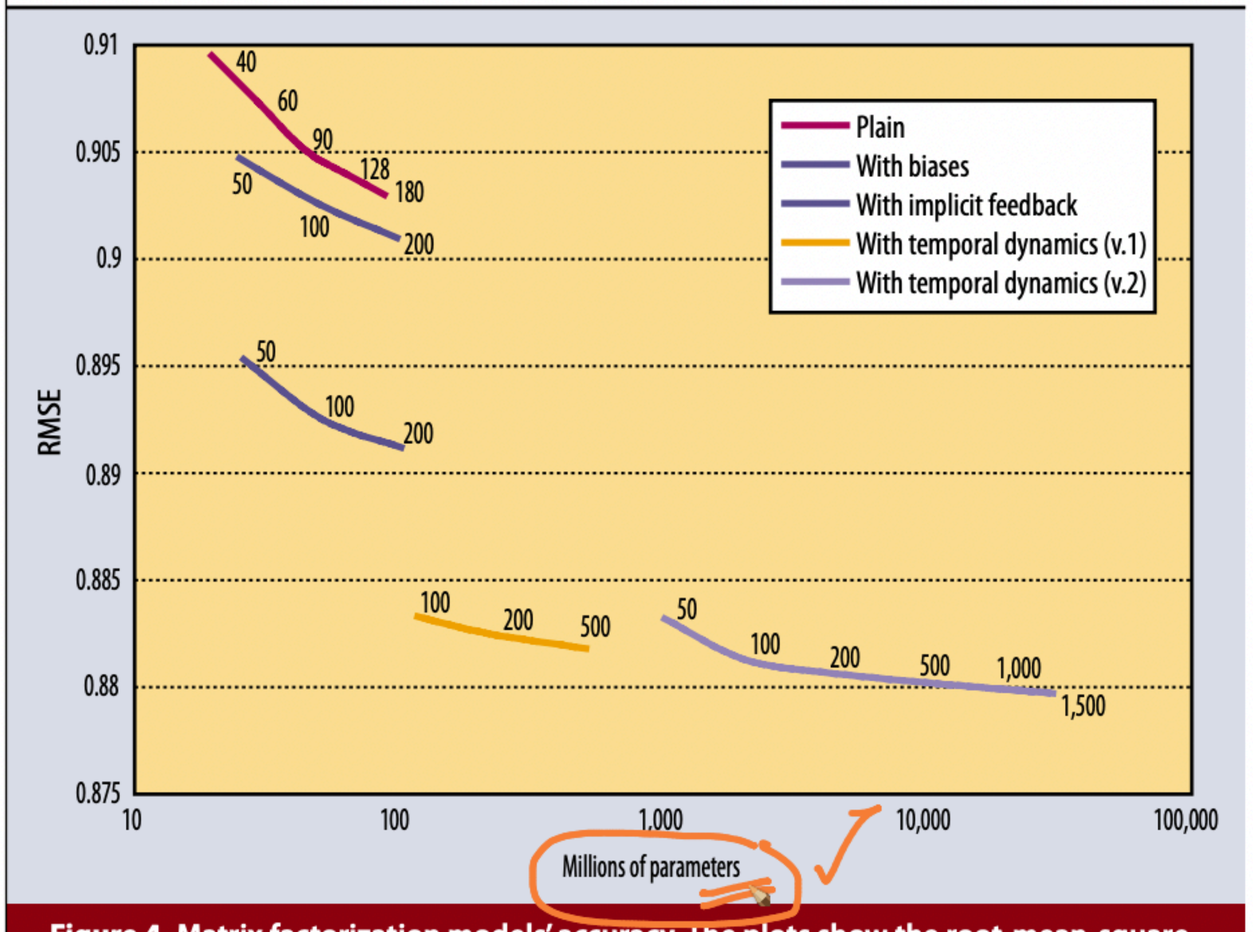


Figure 4. Matrix factorization models' accuracy. The plots show the root-mean-square error of each of four individual factor models (lower is better). Accuracy improves when the factor model's dimensionality (denoted by numbers on the charts) increases. In addition, the more refined factor models, whose descriptions involve more distinct sets of parameters, are more a[illegible]. For comparison, the Netflix system achieves RMSE = 0.9514 on the same d[illegible] RMSE = 0.8563.

Matrix factorization techniques have become a dominant methodology within collaborative filtering recommenders. Experience with datasets such as the Netflix Prize data has shown that they deliver accuracy superior to classical nearest-neighbor techniques. At the same time, they offer a compact memory-efficient model that systems can learn relatively easily. What makes these techniques even more convenient is that models can integrate naturally many crucial aspects of the data, such as multiple forms of feedback, temporal dynamics, and confidence levels.

References

1. D. Goldberg et al., "Using Col[illegible]tive Filtering to Weave [illegible]formation Tapestry,"

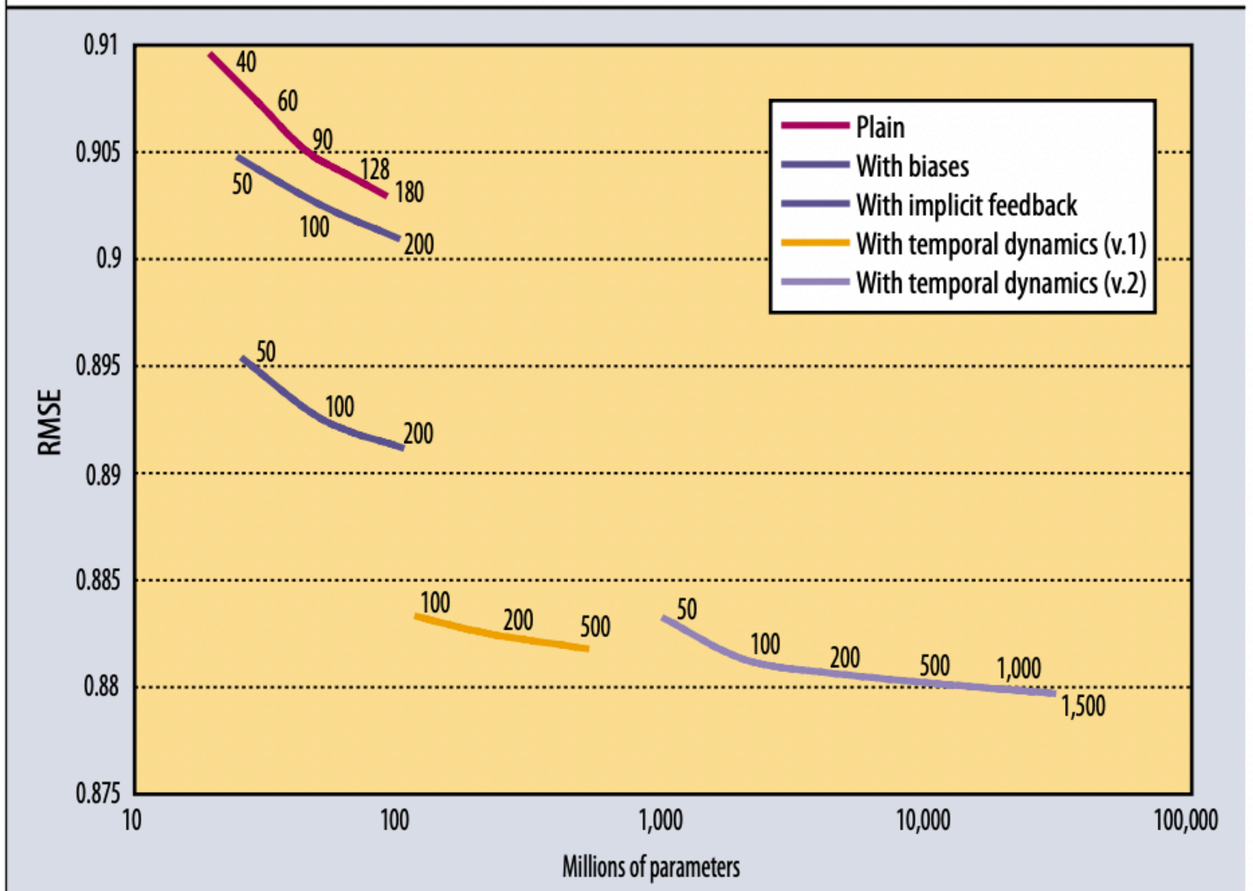


Figure 4. Matrix factorization models' accuracy. The plots show the root-mean-square error of each of four individual factor models (lower is better). Accuracy improves when the factor model's dimensionality (denoted by numbers on the charts) increases. In addition, the more refined factor models, whose descriptions involve more distinct sets of parameters, are more a... For comparison, the Netflix system achieves RMSE = 0.9514 on the same d... RMSE = 0.8563

Matrix factorization techniques have become a dominant methodology within collaborative filtering recommenders. Experience with datasets such as the Netflix Prize data has shown that they deliver accuracy superior to classical nearest-neighbor techniques. At the same time, they offer a compact memory-efficient model that systems can learn relatively easily. What makes these techniques even more convenient is that models can integrate naturally many crucial aspects of the data, such as multiple forms of feedback, temporal dynamics, and confidence levels.

References

1. D. Goldberg et al., "Using Col... tive Filtering to Weave ... Information Tapestry,"

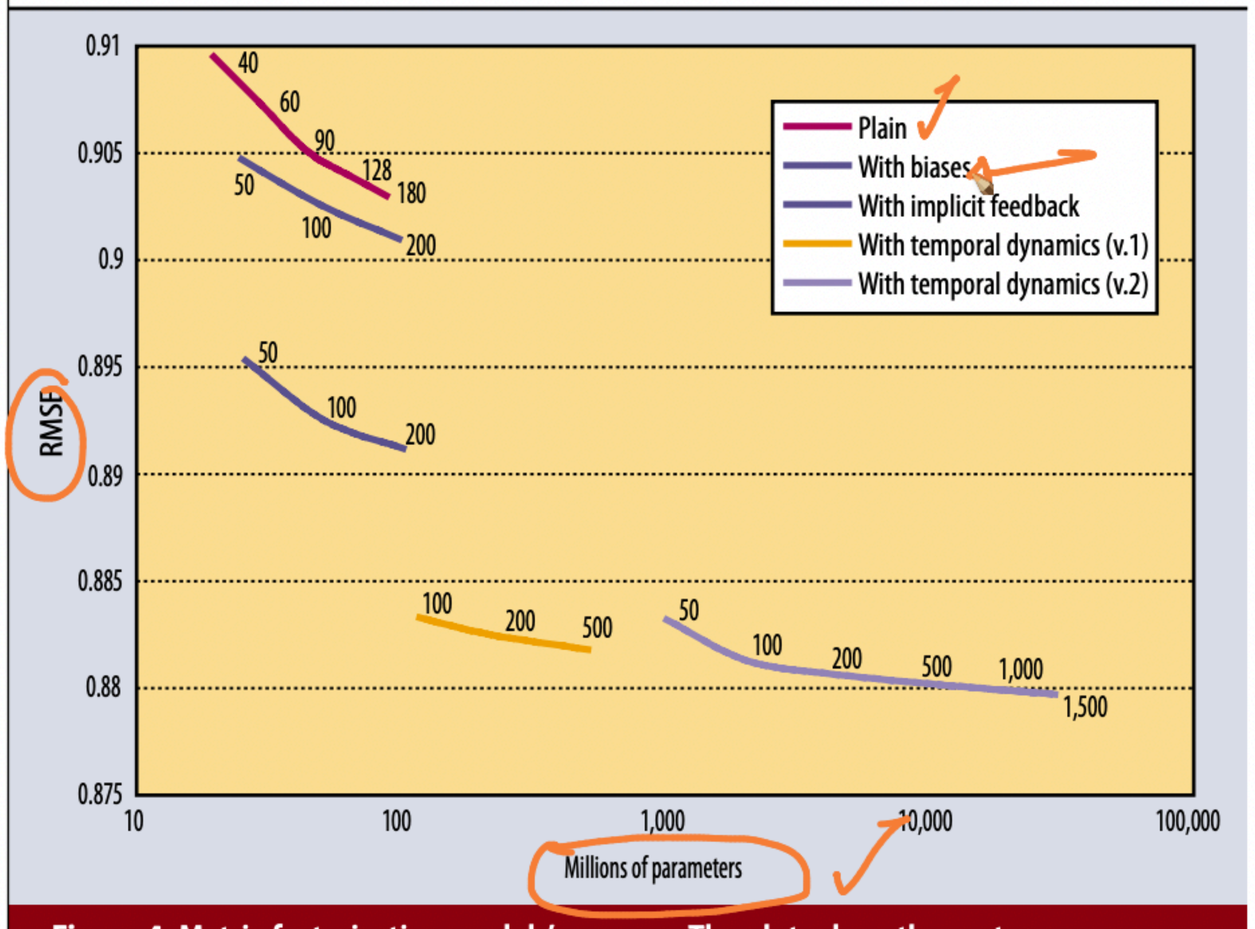


Figure 4. Matrix factorization models' accuracy. The plots show the root-mean-square error of each of four individual factor models (lower is better). Accuracy improves when the factor model's dimensionality (denoted by numbers on the charts) increases. In addition, the more refined factor models, whose descriptions involve more distinct sets of parameters, are more a… RMSE = 0.9514 on the same d… RMSE = 0.8563

Matrix factorization techniques have become a dominant methodology within collaborative filtering recommenders. Experience with datasets such as the Netflix Prize data has shown that they deliver accuracy superior to classical nearest-neighbor techniques. At the same time, they offer a compact memory-efficient model that systems can learn relatively easily. What makes these techniques even more convenient is that models can integrate naturally many crucial aspects of the data, such as multiple forms of feedback, temporal dynamics, and confidence levels.

References

1. D. Goldberg et al., "Using Col… tive Filtering to Weave … Information Tapestry,"

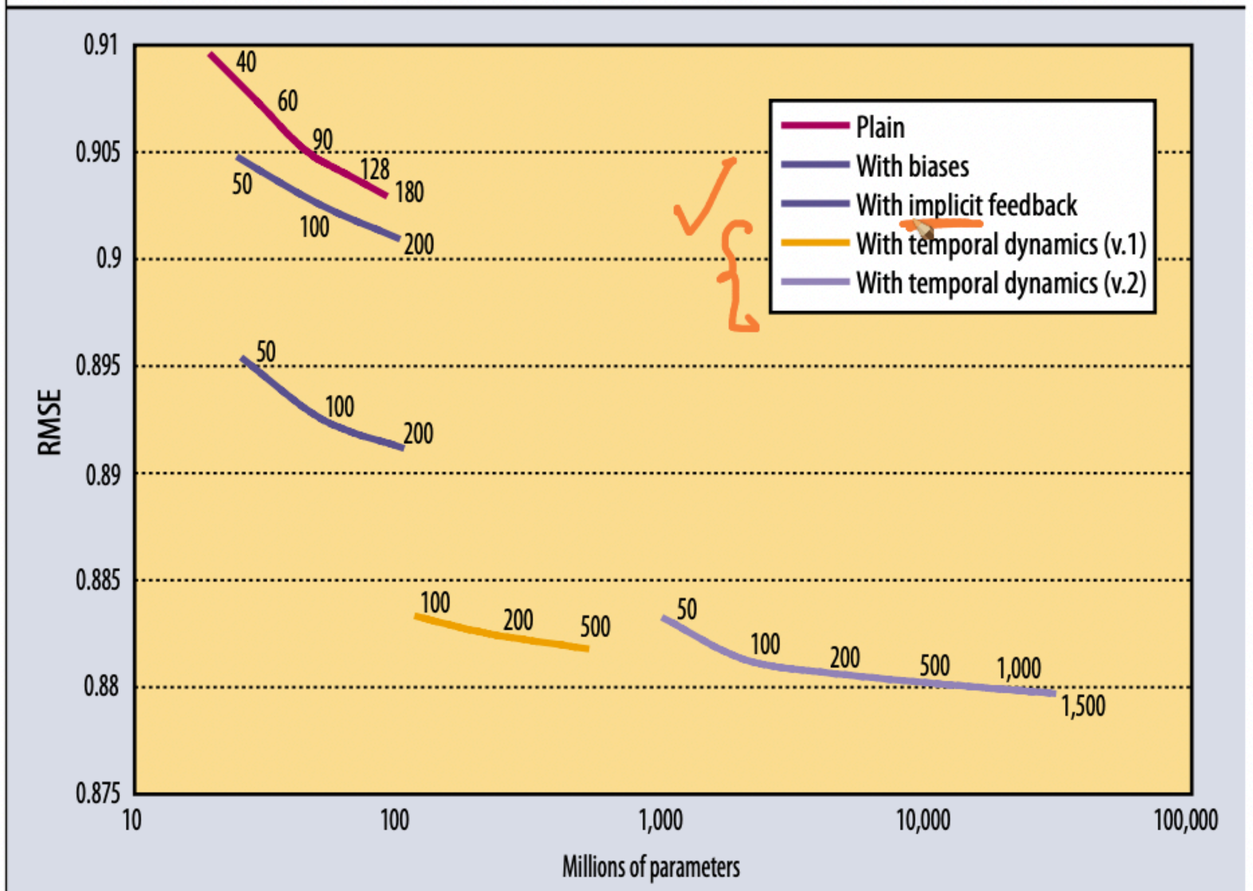


Figure 4. Matrix factorization models' accuracy. The plots show the root-mean-square error of each of four individual factor models (lower is better). Accuracy improves when the factor model's dimensionality (denoted by numbers on the charts) increases. In addition, the more refined factor models, whose descriptions involve more distinct sets of parameters, are more a... For comparison, the Netflix system achieves RMSE = 0.9514 on the same d... RMSE = 0.8563

Matrix factorization techniques have become a dominant methodology within collaborative filtering recommenders. Experience with datasets such as the Netflix Prize data has shown that they deliver accuracy superior to classical nearest-neighbor techniques. At the same time, they offer a compact memory-efficient model that systems can learn relatively easily. What makes these techniques even more convenient is that models can integrate naturally many crucial aspects of the data, such as multiple forms of feedback, temporal dynamics, and confidence levels.

References

1. D. Goldberg et al., "Using Collaborative Filtering to Weave an Information Tapestry,"

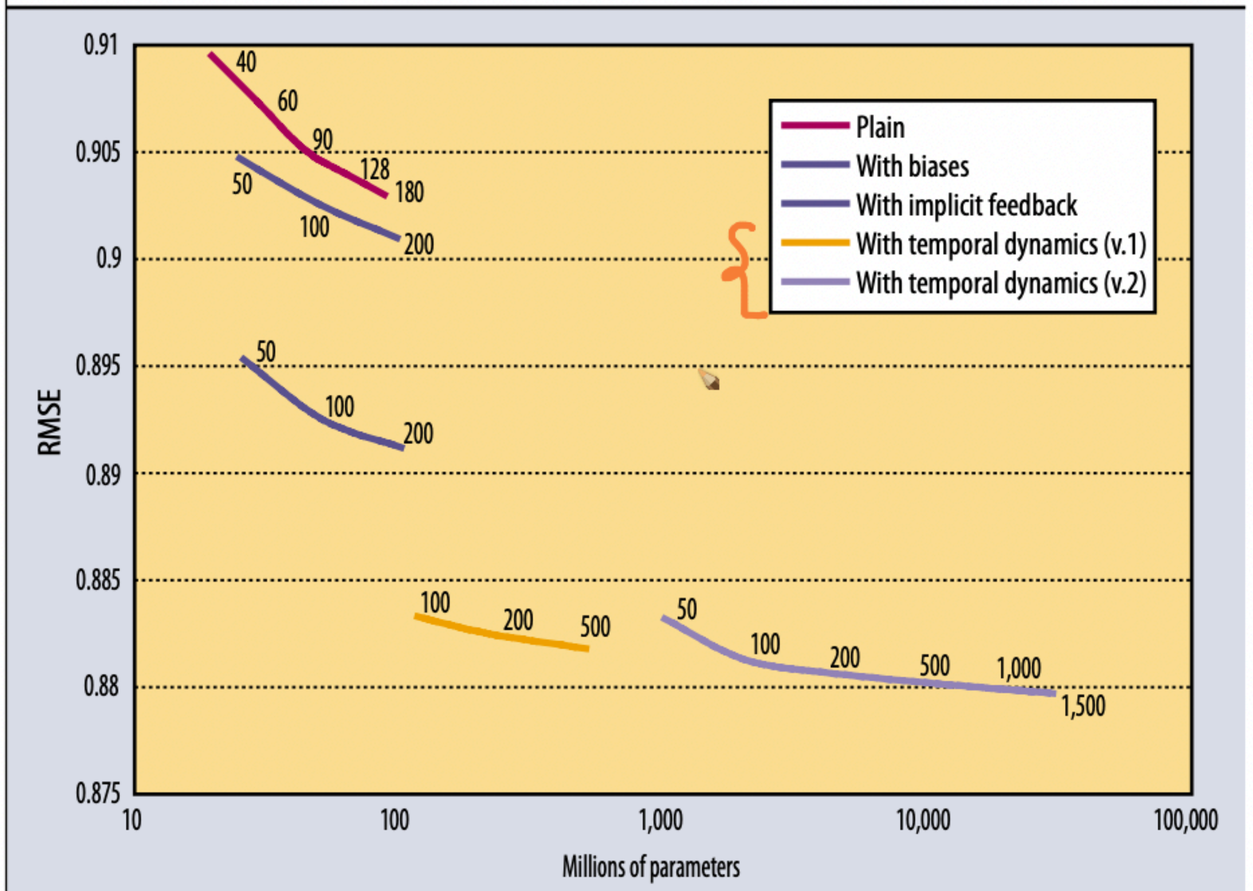


Figure 4. Matrix factorization models' accuracy. The plots show the root-mean-square error of each of four individual factor models (lower is better). Accuracy improves when the factor model's dimensionality (denoted by numbers on the charts) increases. In addition, the more refined factor models, whose descriptions involve more distinct sets of parameters, are more a[illegible]. For comparison, the Netflix system achieves RMSE = 0.9514 on the same d[illegible] RMSE = 0.8563.

Matrix factorization techniques have become a dominant methodology within collaborative filtering recommenders. Experience with datasets such as the Netflix Prize data has shown that they deliver accuracy superior to classical nearest-neighbor techniques. At the same time, they offer a compact memory-efficient model that systems can learn relatively easily. What makes these techniques even more convenient is that models can integrate naturally many crucial aspects of the data, such as multiple forms of feedback, temporal dynamics, and confidence levels. C

References

1. D. Goldberg et al., "Using Col[illegible]tive Filtering to Weave [illegible]nformation Tapestry,"

Temporal dynamics

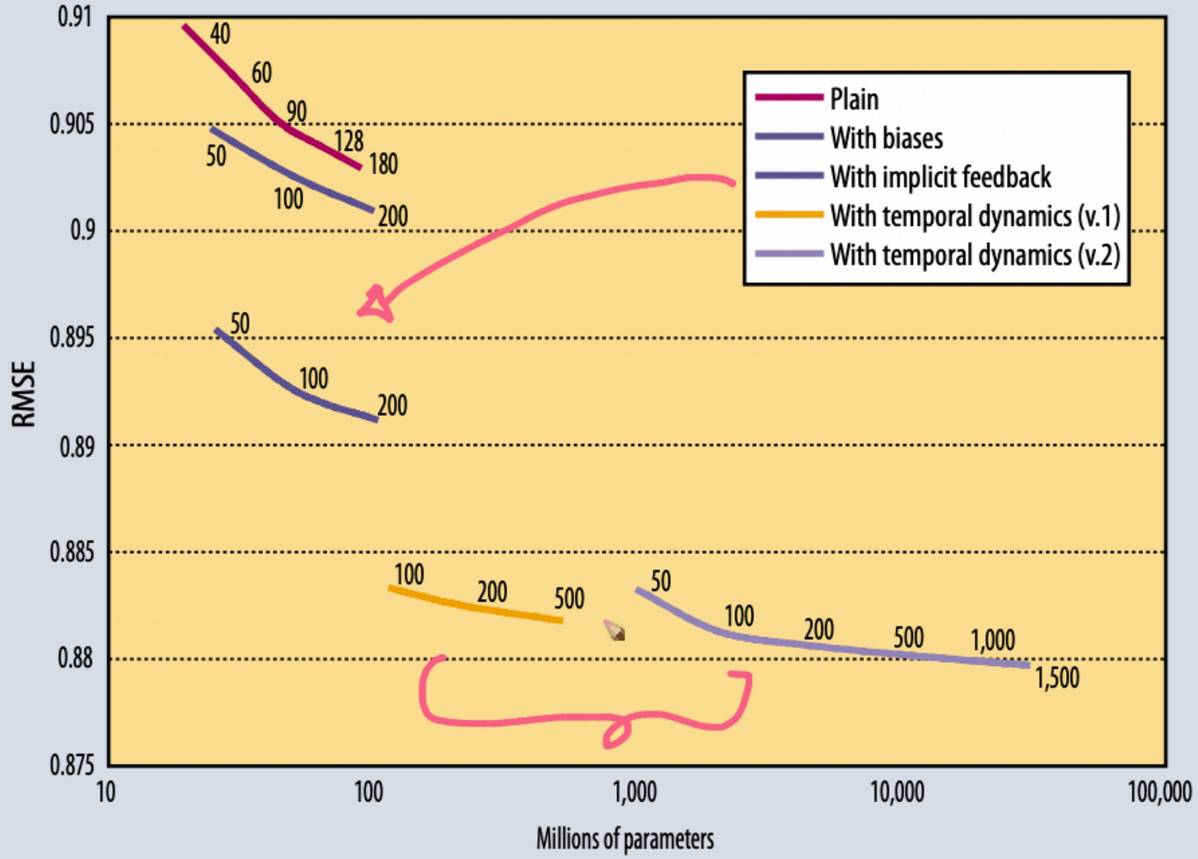


Figure 4. Matrix factorization models' accuracy. The plots show the root-mean-square error of each of four individual factor models (lower is better). Accuracy improves when the factor model's dimensionality (denoted by numbers on the charts) increases. In addition, the more refined factor models, whose descriptions involve more distinct sets of parameters, are more accurate. For comparison, the Netflix system achieves RMSE = 0.9514 on the same d… RMSE = 0.8563.

Matrix factorization techniques have become a dominant methodology within collaborative filtering recommenders. Experience with datasets such as the Netflix Prize data has shown that they deliver accuracy superior to classical nearest-neighbor techniques. At the same time, they offer a compact memory-efficient model that systems can learn relatively easily. What makes these techniques even more convenient is that models can integrate naturally many crucial aspects of the data, such as multiple forms of feedback, temporal dynamics, and confidence levels. ■

References

1. D. Goldberg et al., "Using Col… tive Filtering to Weave …formation Tapestry," Comm. ACM, vol. 35, 1992, pp.

$$|N(u)|^{-0.5} \sum_{i \in N(u)} x_i \,.^{4,5}$$

Another information source is known user attributes, for example, demographics. Again, for simplicity consider Boolean attributes where user u corresponds to the set of attributes $A(u)$, which can describe gender, age group, Zip code, income level, and so on. A distinct factor vector $y_a \in \mathbb{R}^f$ corresponds to each attribute to describe a user through the set of user-associated attributes:

$$\sum_{a \in A(u)} y_a$$

The matrix factorization model should integrate all signal sources, with enhanced user representation:

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T [p_u + |N(u)|^{-0.5} \sum_{i \in N(u)} x_i + \sum_{a \in A(u)} y_a] \quad (6)$$

While the previous examples deal with enhancing user representation—where lack of data is more common—items can get a similar treatment when necessary.

TEMPORAL DYNAMICS

So far, the presented models have been static. In reality, product perception and popularity constantly change as new selections emerge. Similarly, customers' inclinations evolve, leading them to redefine their taste. Thus, the system should account [illegible] frequency of actions, reflecting the dynamic, time- [illegible] user watched a certain might become a fan of crime dramas a year later. Similarly, humans change their perception of certain actors and directors. The model accounts for this effect by taking the user factors (the vector p_u) as a function of time. On the other hand, it specifies static item characteristics, q_i, because, unlike humans, items are static in nature.

Exact parameterizations of time-varying parameters[11] lead to replacing Equation 4 with the dynamic prediction rule for a rating at time t:

$$\hat{r}_{ui}(t) = \mu + b_i(t) + b_u(t) + q_i^T p_u(t) \quad (7)$$

INPUTS WITH VARYING CONFIDENCE LEVELS

In several setups, not all observed ratings deserve the same weight or confidence. For example, massive advertising might influence votes for certain items, which do not aptly reflect longer-term characteristics. Similarly, a system might face adversarial users that try to tilt the ratings of certain items.

Another example is systems built around implicit feedback. In such systems, which interpret ongoing user behavior, a user's exact preference level is hard to quantify. Thus, the system works with a cruder binary representation, stating either "probably likes the product" or "probably not interested in the product." In such cases, it is valuable to attach confidence scores with the estimated preferences. Confidence can stem from available

$$|N(u)|^{-0.5} \sum_{i \in N(u)} x_i \,.^{4,5}$$

Another information source is known user attributes, for example, demographics. Again, for simplicity consider Boolean attributes where user u corresponds to the set of attributes $A(u)$, which can describe gender, age group, Zip code, income level, and so on. A distinct factor vector $y_a \in \mathbb{R}^f$ corresponds to each attribute to describe a user through the set of user-associated attributes:

$$\sum_{a \in A(u)} y_a$$

The matrix factorization model should integrate all signal sources, with enhanced user representation:

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T [p_u + |N(u)|^{-0.5} \sum_{i \in N(u)} x_i + \sum_{a \in A(u)} y_a] \quad (6)$$

While the previous examples deal with enhancing user representation—where lack of data is more common—items can get a similar treatment when necessary.

TEMPORAL DYNAMICS

So far, the presented models have been static. In reality, product perception and popularity constantly change as new selections emerge. Similarly, customers' inclinations evolve, leading them to redefine their taste. Thus, the system should account [illegible] frequency of actions, reflecting the dynamic, time- [illegible] user watched a certain might become a fan of crime dramas a year later. Similarly, humans change their perception of certain actors and directors. The model accounts for this effect by taking the user factors (the vector p_u) as a function of time. On the other hand, it specifies static item characteristics, q_i, because, unlike humans, items are static in nature.

Exact parameterizations of time-varying parameters[11] lead to replacing Equation 4 with the dynamic prediction rule for a rating at time t:

$$\hat{r}_{ui}(t) = \mu + b_i(t) + b_u(t) + q_i^T p_u(t) \quad (7)$$

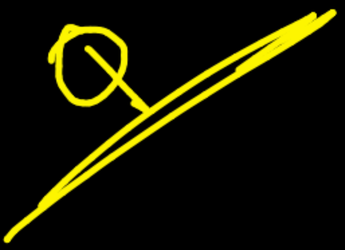
INPUTS WITH VARYING CONFIDENCE LEVELS

In several setups, not all observed ratings deserve the same weight or confidence. For example, massive advertising might influence votes for certain items, which do not aptly reflect longer-term characteristics. Similarly, a system might face adversarial users that try to tilt the ratings of certain items.

Another example is systems built around implicit feedback. In such systems, which interpret ongoing user behavior, a user's exact preference level is hard to quantify. Thus, the system works with a cruder binary representation, stating either "probably likes the product" or "probably not interested in the product." In such cases, it is valuable to attach confidence scores with the estimated preferences. Confidence can stem from available

μ: const ✓

{ b_u, b_i ; q_i ; p_u } time-varying → gave very high reduction in RMSE



Youtube

recency aware time-based RecSys

→ update models : b_u, b_i, q_i, P_u periodically

→ more weightage to more recent data

↳ ✓ add a w_{ui} based on recency

Simpler

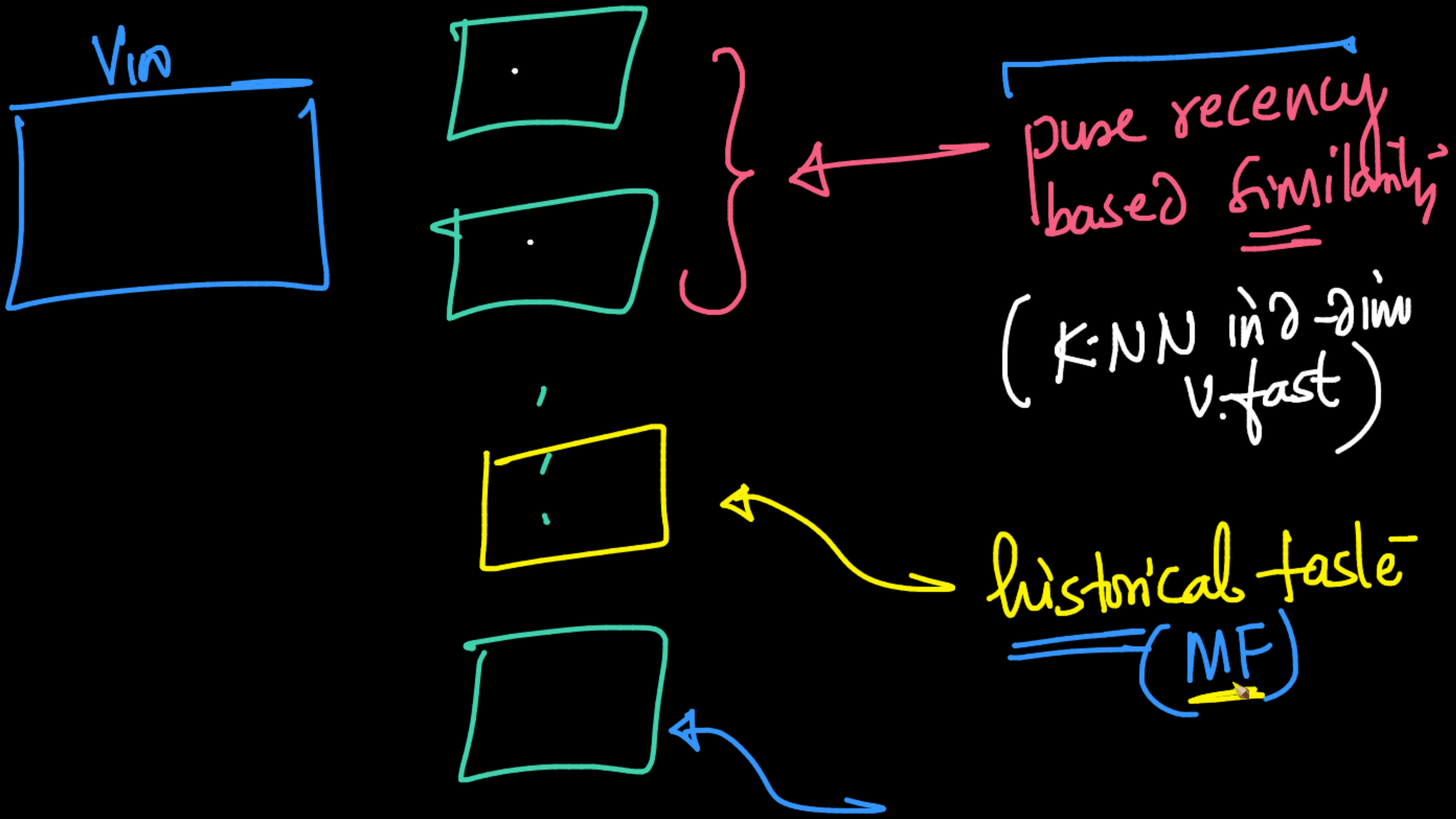
Nightly

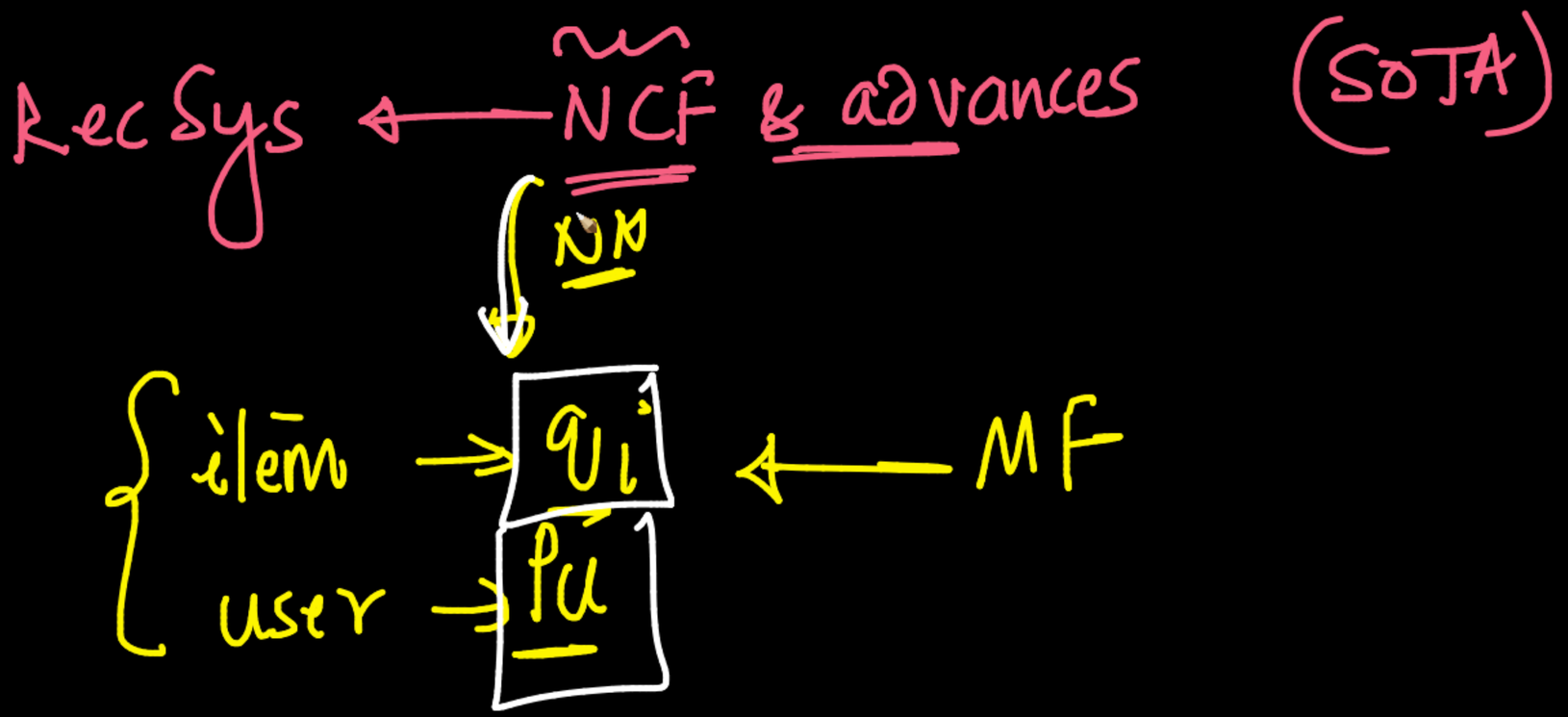
$r_{ui} \longrightarrow MF\left(q_i, b_i\right)$

$q_i \in \mathbb{R}^d$ → dim-rep of i

$b_i \approx$ avg. rating of i → popularity

Must → {top 100 videos that are good to recommend




RecSys
NCF & advances
(SOTA)
NN
item
user
q_i
p_u
MF

```
  Building wheel for scikit-surprise (setup.py) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.1-cp37-cp37m-linux_x86_64.whl size
  Stored in directory: /root/.cache/pip/wheels/76/44/74/b498c42be47b2406bd27994e16c5188e337c657025ab
Successfully built scikit-surprise
Installing collected packages: scikit-surprise, surprise
Successfully installed scikit-surprise-1.1.1 surprise-0.1
```

```
#Source: https://surprise.readthedocs.io/en/stable/getting_started.html
from surprise import SVD
from surprise import Dataset
from surprise.model_selection import train_test_split
from surprise import accuracy
```

```
# Load the movielens-100k dataset (download it if needed),
data = Dataset.load_builtin('ml-100k')
```

```
# sample random trainset and testset
# test set is made of 25% of the ratings.
```

MovieLens 100K Dataset

MovieLens 100K movie ratings. Stable benchmark dataset. 100,000 ratings from 1000 users on 1700 movies.
Released 4/1998.

- README.txt
- ml-100k.zip (size: 5 MB, checksum)
- Index of unzipped files

Permalink: https://grouplens.org/datasets/movielens/100k/

Datasets

MovieLens 100K Dataset

10^5

MovieLens 100K movie ratings. Stable benchmark dataset. 100,000 ratings from 1000 users on 1700 movies.
Released 4/1998.

10^3 1.7×10^3

- README.txt
- ml-100k.zip (size: 5 MB, checksum)
- Index of unzipped files

Permalink: https://grouplens.org/datasets/movielens/100k/

1.7×10^6

Datasets

- MovieLens
- WikiLens
- Book-Crossing
- Book Genome Dataset
- Jester
- EachMovie
- HetRec 2011
- Serendipity 2018
- Personality 2018
- Learning from Sets of Items 2019

```
# Load the movielens-100k dataset (download it if needed),
data = Dataset.load_builtin('ml-100k')
```

```
# sample random trainset and testset
# test set is made of 25% of the ratings.
trainset, testset = train_test_split(data, test_size=.25)

# We'll use the famous SVD algorithm.
algo = SVD()

# Train the algorithm on the trainset, and predict ratings for the testset
algo.fit(trainset)
predictions = algo.test(testset)

# Then compute RMSE
accuracy.rmse(predictions)
```

```
RMSE: 0.9463
0.9462710000722717
```

```
# Load the movielens-100k dataset (download it if needed),
data = Dataset.load_builtin('ml-100k')
```

```
# sample random trainset and testset
# test set is made of 25% of the ratings.
trainset, testset = train_test_split(data, test_size=.25)

# We'll use the famous SVD algorithm.
algo = SVD()

# Train the algorithm on the trainset, and predict ratings for the testset
algo.fit(trainset)
predictions = algo.test(testset)

# Then compute RMSE
accuracy.rmse(predictions)
```

```
RMSE: 0.9463
0.9462710000722717
```

```
# test set is made of 25% of the ratings.
trainset, testset = train_test_split(data, test_size=.25)

# We'll use the famous SVD algorithm.
algo = SVD()

# Train the algorithm on the trainset, and predict ratings for the testset
algo.fit(trainset)
predictions = algo.test(testset)

# Then compute RMSE
accuracy.rmse(predictions)
```

```
RMSE: 0.9463
0.9462710000722717
```

```
[ ] uid = str(196)  # raw user id (as in the ratings file). They are **strings**!
    iid = str(302)  # raw item id (as in the ratings file). They are **strings**!

    # get a prediction for specific users and items.
    pred = algo.predict(uid, iid)
    pred
```

```
algo.fit(trainset)
predictions = algo.test(testset)

# Then compute RMSE
accuracy.rmse(predictions)
```

```
RMSE: 0.9463
0.9462710000722717
```

```
uid = str(196)  # raw user id (as in the ratings file). They are **strings**!
iid = str(302)  # raw item id (as in the ratings file). They are **strings**!

# get a prediction for specific users and items.
pred = algo.predict(uid, iid)
pred
```

```
Prediction(uid='196', iid='302', r_ui=None, est=4.227320528336027, details={'was_impossible':
False})
```

Using prediction algorithms

Surprise provides a bunch of built-in algorithms. All algorithms derive from the `AlgoBase` base class, where are implemented some key methods (e.g. `predict`, `fit` and `test`). The list and details of the available prediction algorithms can be found in the `prediction_algorithms` package documentation.

Every algorithm is part of the global Surprise namespace, so you only need to import their names from the Surprise package, for example:

```
from surprise import KNNBasic
algo = KNNBasic()
```

Some of these algorithms may use baseline estimates, some may use a similarity measure. We will here review how to configure the way baselines and similarities are computed.

Baselines estimates configuration

Note

This section only applies to algorithms (or similarity measures) that try to minimize the following regularized squared error (or equivalent):

$$\sum_{r_{ui} \in R_{train}} (r_{ui} - (\mu + b_u + b_i))^2 + \lambda \left(b_u^2 + b_i^2\right) .$$

For algorithms using

MF → surprise

Q

✓ { 10's of millions of params

deep-learning

✓ desktop / laptop → colab ✓

→ NMF & MF → Spark (MLops)

class `sklearn.decomposition.`**`NMF`**(*n_components=None, *, init=None, solver='cd', beta_loss='frobenius', tol=0.0001, max_iter=200, random_state=None, alpha='deprecated', alpha_W=0.0, alpha_H='same', l1_ratio=0.0, verbose=0, shuffle=False, regularization='deprecated'*) [source]

Non-Negative Matrix Factorization (NMF).

Find two non-negative matrices, i.e. matrices with all non-negative elements, (W, H) whose product approximates the non-negative matrix X. This factorization can be used for example for dimensionality reduction, source separation or topic extraction.

The objective function is:

$$
\begin{aligned}
L(W, H) = & \; 0.5 * ||X - WH||_{loss}^2 \\
& + alpha\_W * l1\_ratio * n\_features * ||vec(W)||_1 \\
& + alpha\_H * l1\_ratio * n\_samples * ||vec(H)||_1 \\
& + 0.5 * alpha\_W * (1 - l1\_ratio) * n\_features * ||W||_{Fro}^2 \\
& + 0.5 * alpha\_H * (1 - l1\_ratio) * n\_samples * ||H||_{Fro}^2
\end{aligned}
$$

Where:

$||A||_{Fro}^2 = \sum_{i,j} A_{ij}^2$ (Frobenius norm)

$||vec(A)||_1 = \sum_{i,j} abs(A_{ij})$ (Elementwise L1 norm)

The generic nor… …upported beta-divergence loss. The choice

Entities

$\in \mathbb{R}^d$

q_i . p_u

Classfn, reg, Viz; Clustg

we caused the most-

MF: feature Engineering

using a model

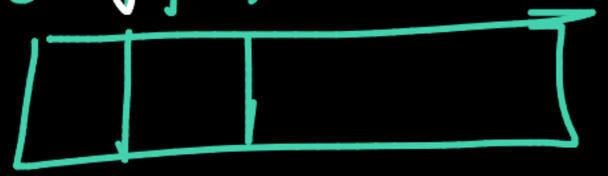
γ_{ui}

1 User, Movie

$d = 10$

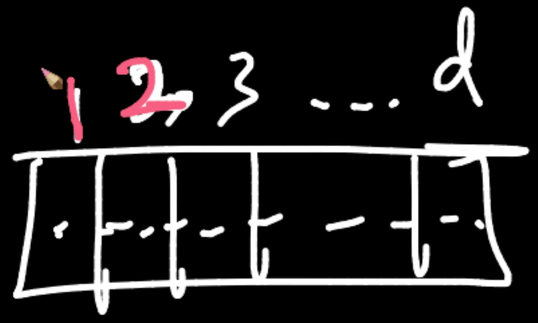
manual FE

user: County Pin . . .



MF . FE

user:



no interpretability

→ Viz (UMAP / tSNE)

d.

d.

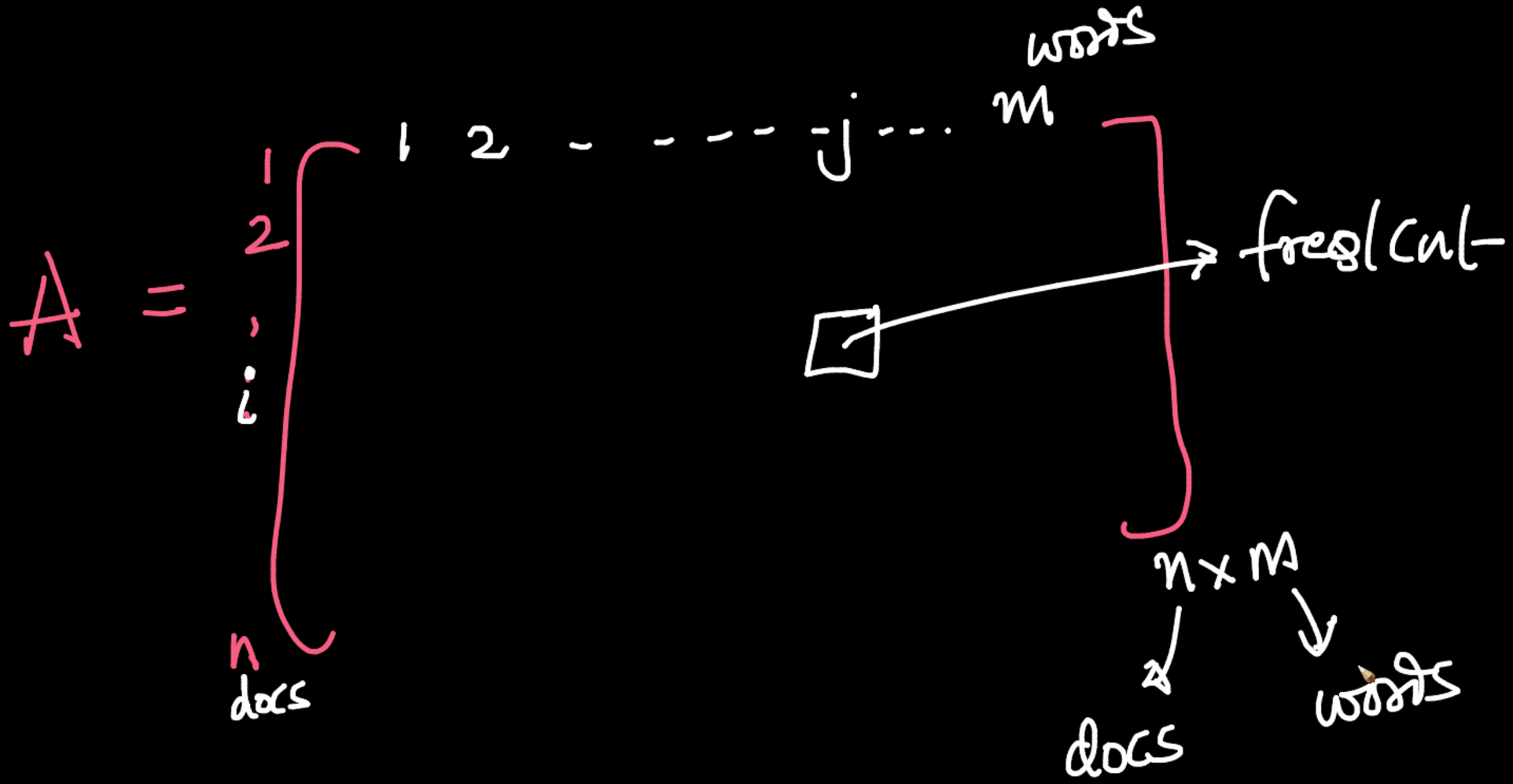
Text

MF

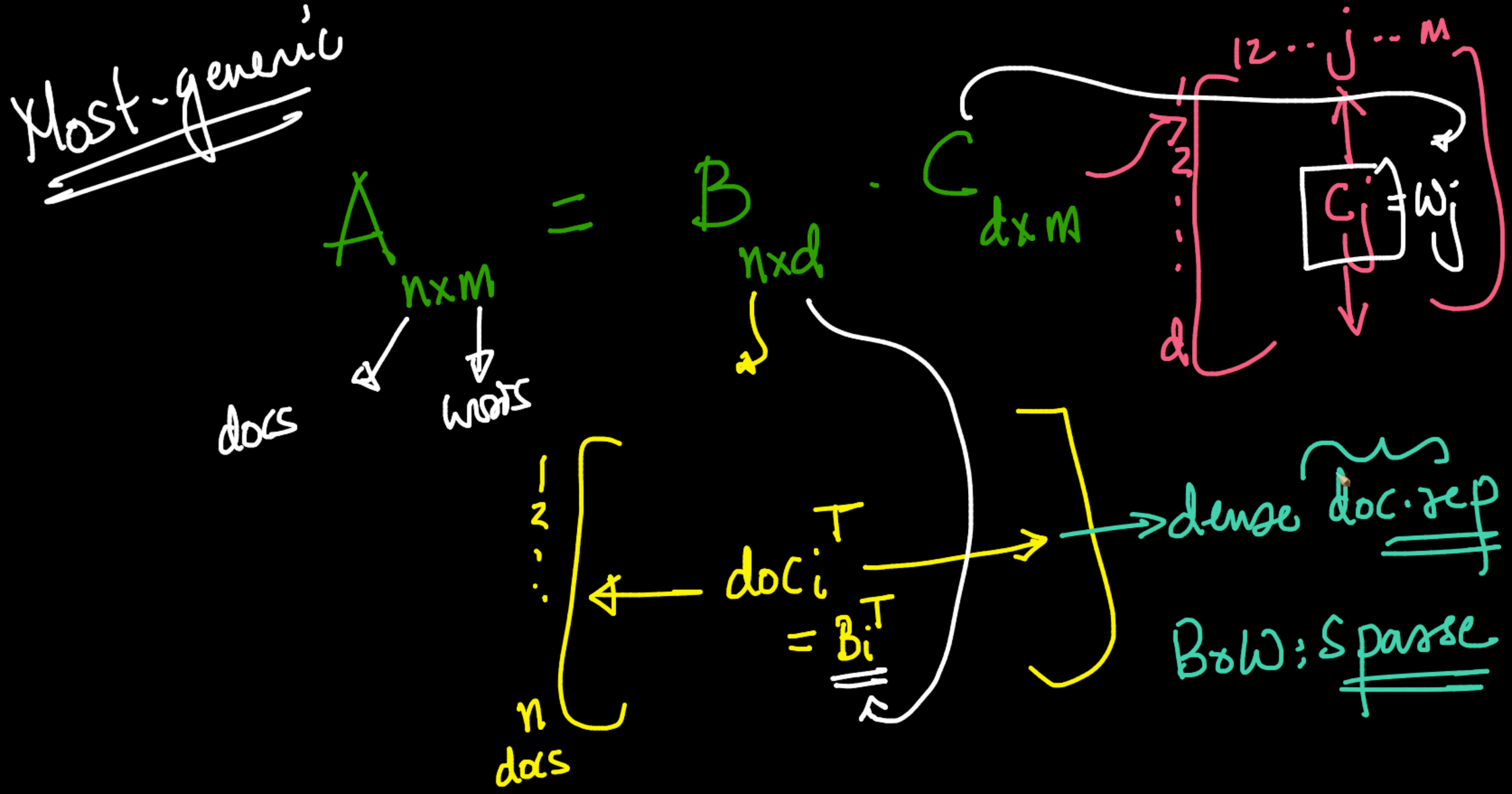
d-dim for words → d-dim → Sensible

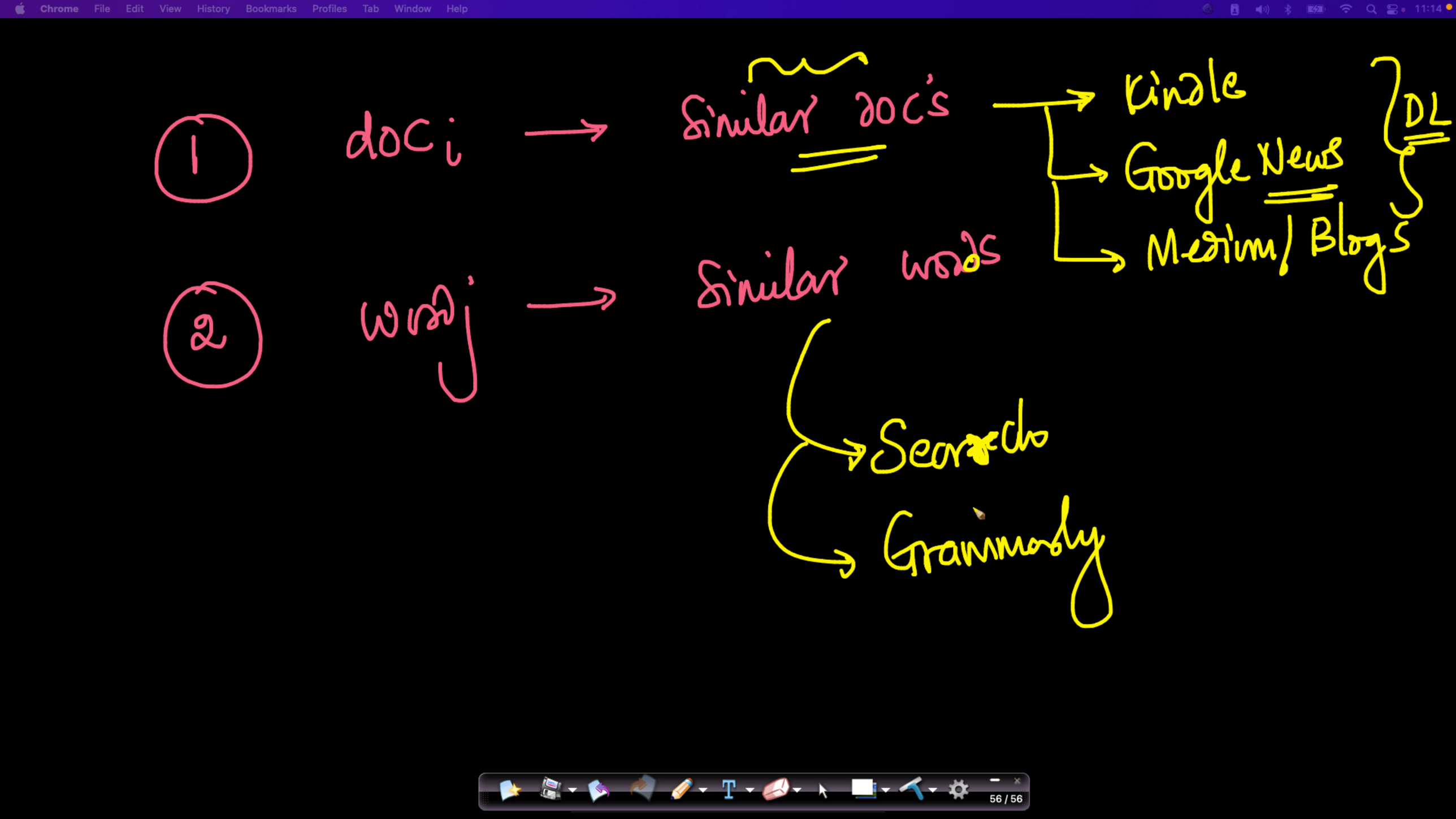
need not be interpretability

Large Corpus of text → Wikipedia Dataset


words
1 2 - - - - j - - - m
A =
1
2
i
n
docs
freq(w,d)
n x m
docs
words

Most-generic




1
doc i
Similar docs
Kindle
Google News
Medium / Blogs
DL
2
word
Similar words
Search
Grammarly

Alt

$X_{m \times m}$ =

	1	2	-	-	-	j	-	-	m words
1									
2									
:									
i									
:									
m words									

m x m

$X_{ij} = 1$ if w_i & w_j occur in particular window context

See also:

DictionaryLearning
Find a dictionary that sparsely encodes data.

MiniBatchSparsePCA
Mini-batch Sparse Principal Components Analysis.

PCA
Principal component analysis.

SparseCoder
Find a sparse representation of data from a fixed, precomputed dictionary.

SparsePCA
Sparse Principal Components Analysis.

TruncatedSVD
Dimensionality reduction using truncated SVD.

Context

References

[1]
"Fast local algorithms for large scale nonnegative matrix and tensor factorizations" Cichocki, Andrzej, and P. H. A. N. Anh-Huy. IEICE transactions on fundamentals of electronics, communications and computer sciences 92.3: 708-721, 2009.

[2]
"Algorithms for nonnegative matrix factorization with the beta-divergence" Fevotte, C., & Idier, J. (2011). Neural Computation, 23(9).

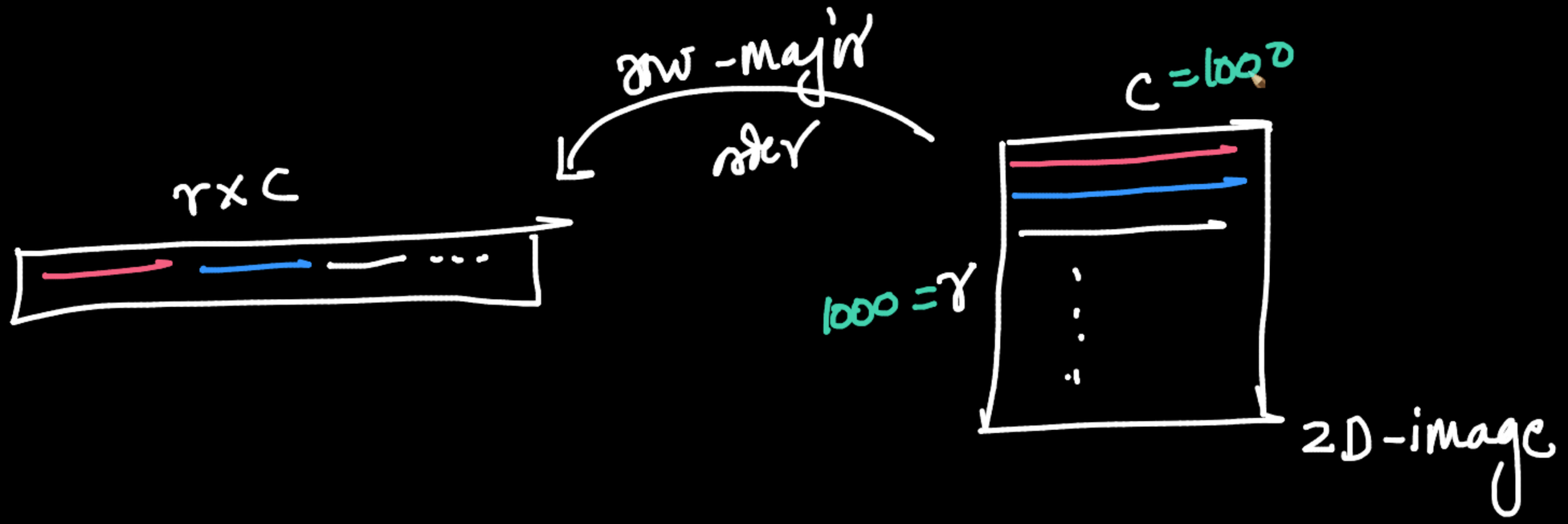
Examples

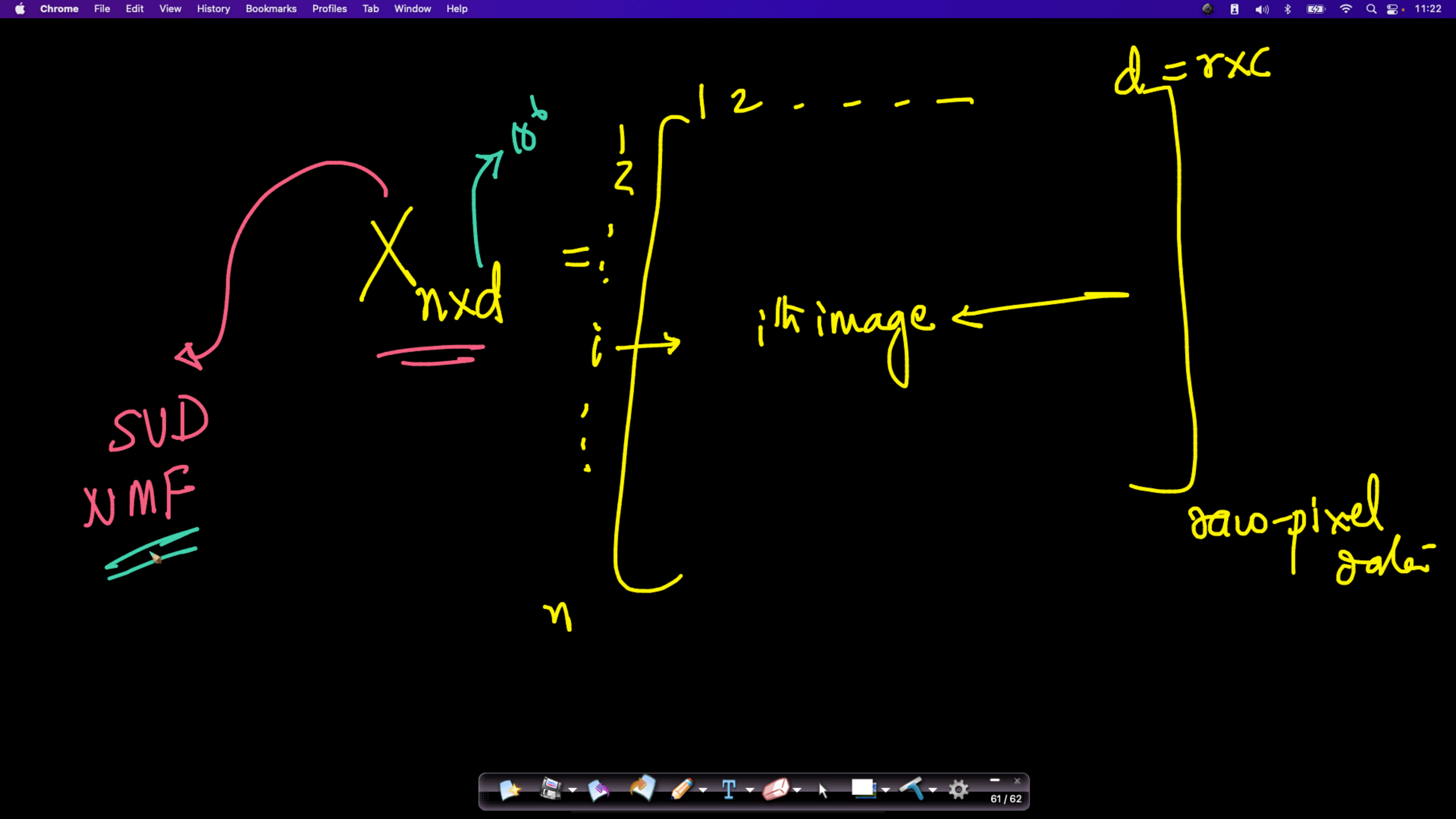
$$X_{n \times n} = B_{n \times d} \; C_{d \times n}$$

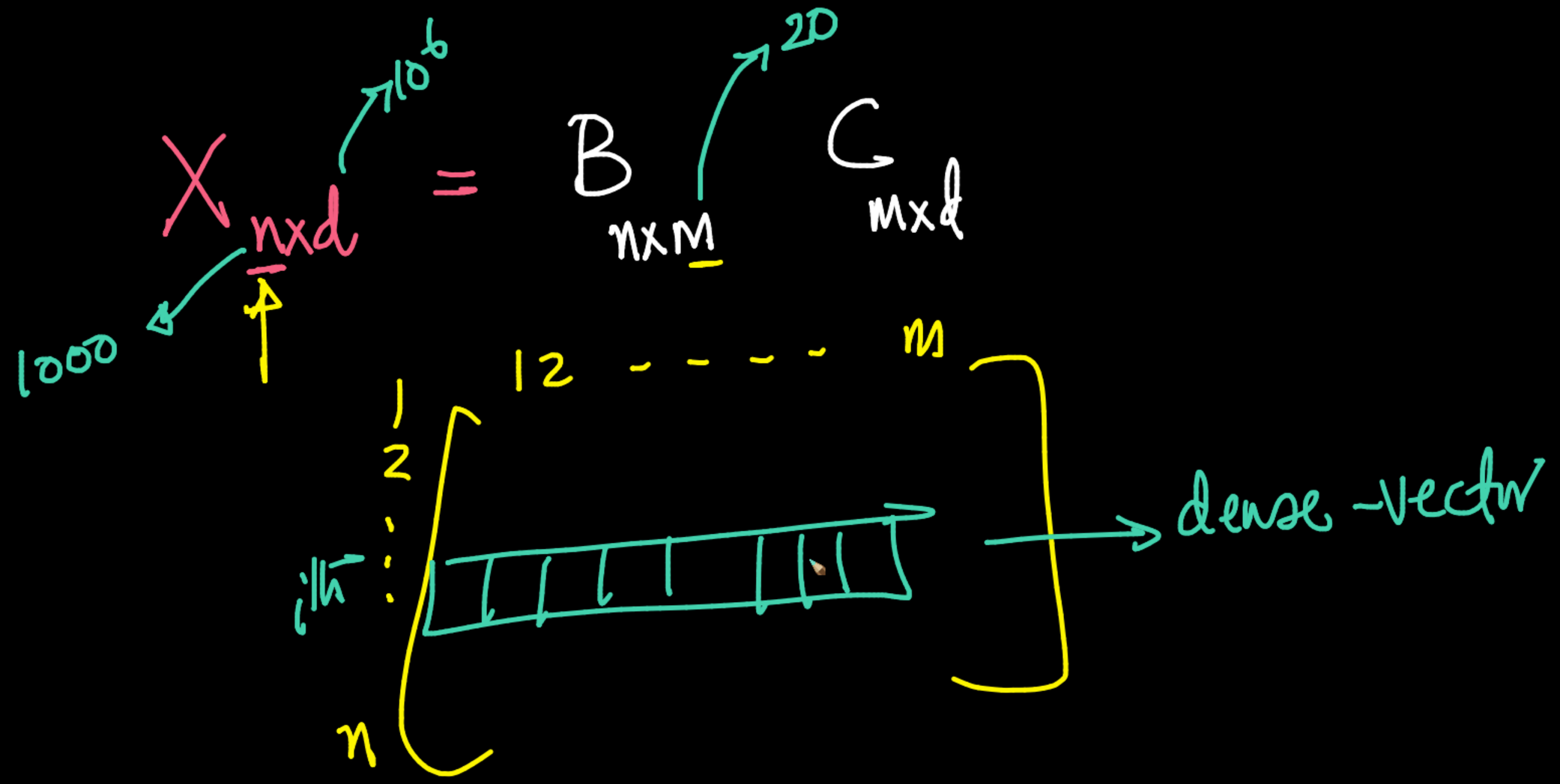
B_i

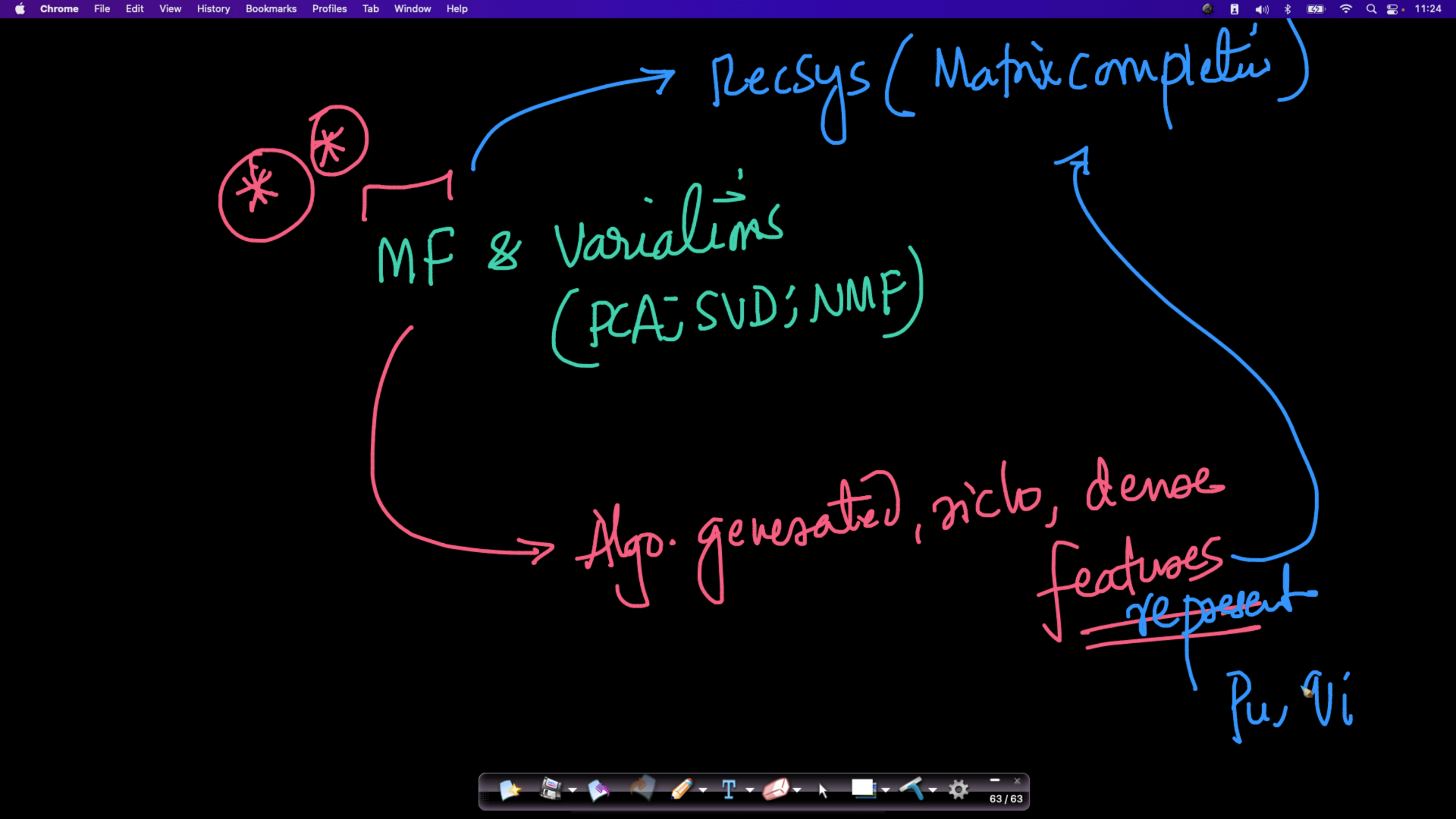
C_j

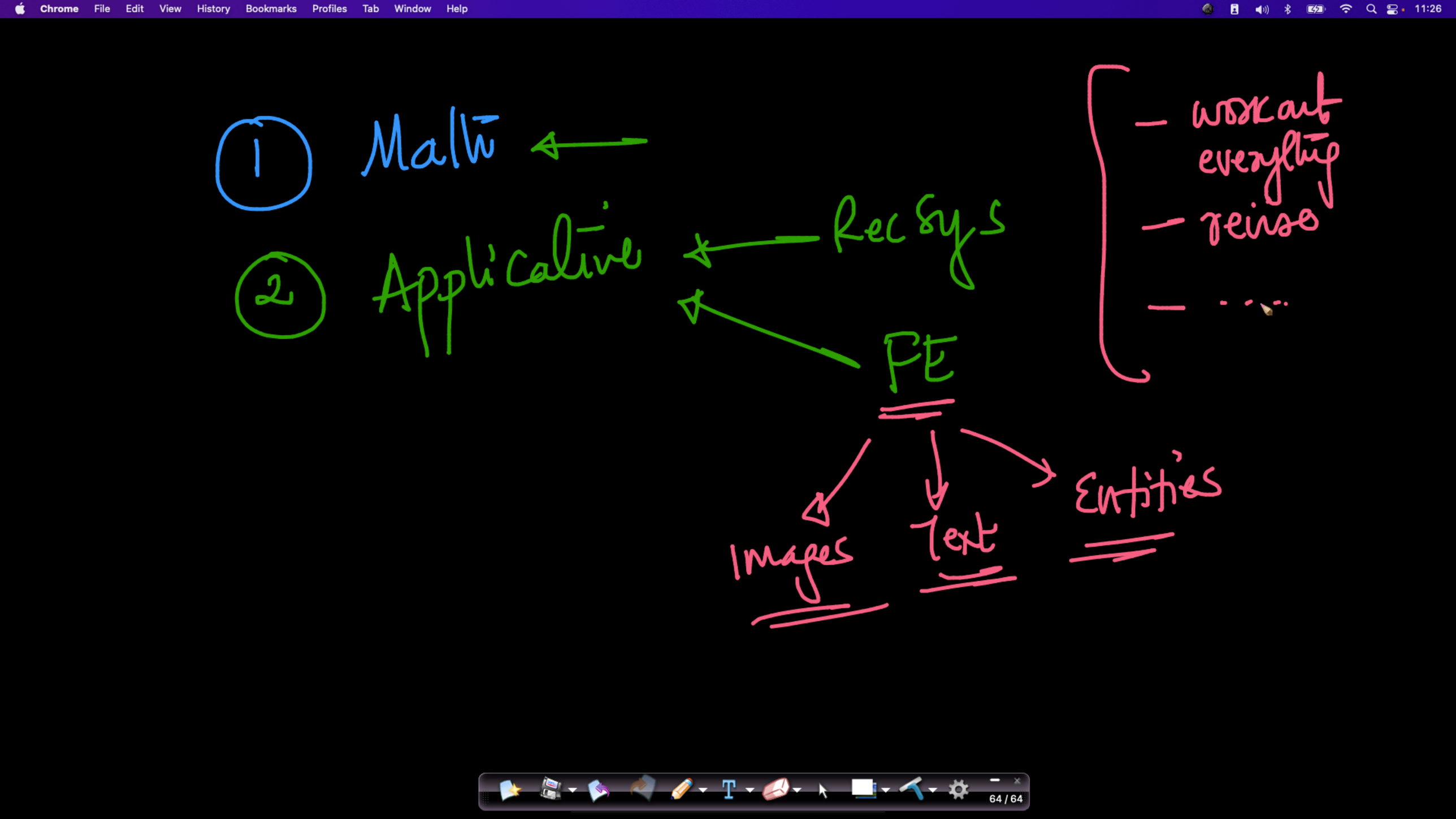
Images

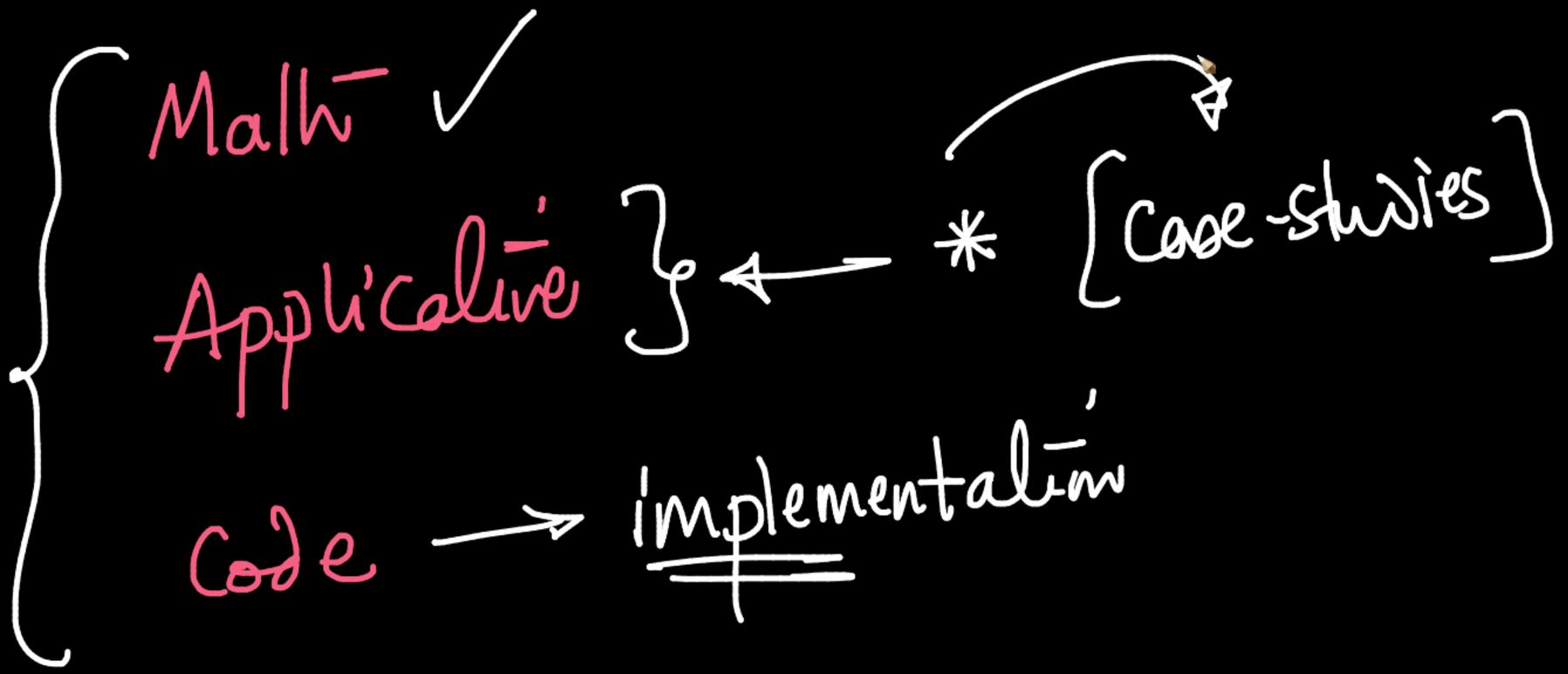



Chrome File Edit View History Bookmarks Profiles Tab Window Help
11:22
$d = r \times c$
$X_{n \times d}$
to be
SVD
NMF
1 2 - - - - -
1
2
i
ith image
n
gray-pixel data
61 / 62



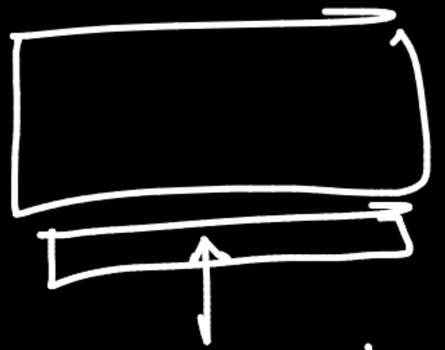

Chrome File Edit View History Bookmarks Profiles Tab Window Help
11:24
RecSys (Matrix completion)
MF & Variations
(PCA; SVD; NMF)
Algo. generated, sicko, dense features
represent
Pu, Qi
63 / 63

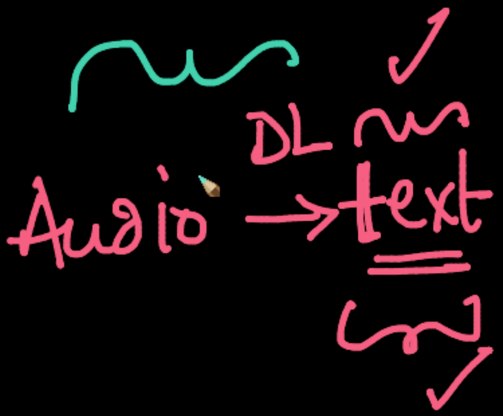

1
Multi
2
Applicative
RecSys
PE
- workout everything
- reinso
- ...
Images
Text
Entities



Search - kw

- webpage → kw
- video, ts → kw





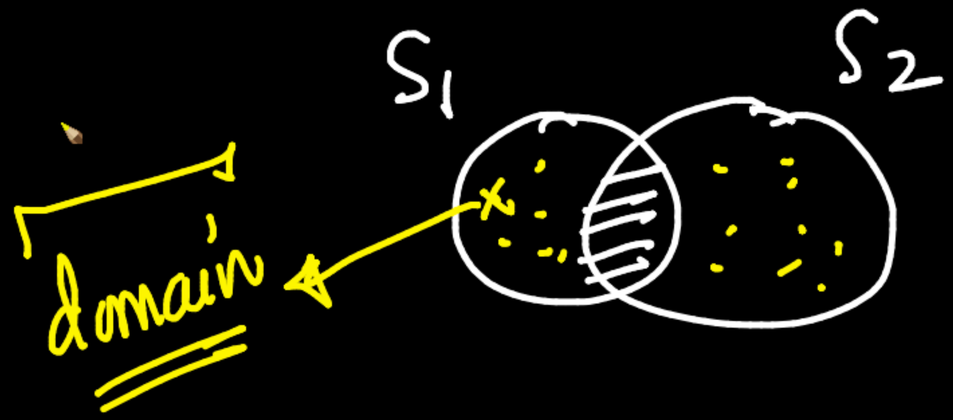
Q

outliers

S_1 ← Isolation Forest

S_2 ← LOF

ground truth — NOT given



Chrome File Edit View History Bookmarks Profiles Tab Window Help 11:43

Using prediction algor | RecSys_MF.ipynb - C | MovieLens 100K Data | Evaluation of outlier d | Recommender-Systen | Microsoft Word - Bell | sklearn.decompositior | matrix factorization - | New Tab

Search Google or type a URL

Gmail Images

InterviewBit Software Services | Live | ML, RS-3 (Basket Analysi

scaler.com/meetings/i/ml-rs-3-basket-analysis-association-rules/live

You have left the meeting

We get frequent requests for your notes!

Notes written by you helps in understanding the topic better. You can upload the notes in two simple steps mentioned below

1. **Scan the QR code with your iPad**
   Scanner should be present in the top menu on your iPad
2. **Upload Notes on the generated link**
   All notes uploaded will be visible in the saved version of this session

OR

Drag and drop files or **click here to upload**

Files Uploaded from your computer appear here

Photo by NASA Image Library

68 / 68