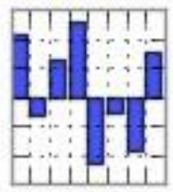
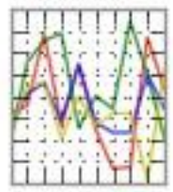
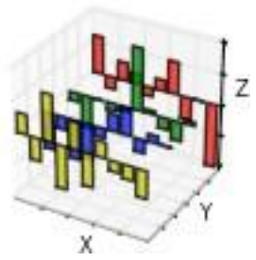
pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$







Maik Röder

Python Barcelona Meetup

7. February 2013

Python Consultant

maikroeder@gmail.com

Pandas

- Powerful and productive Python data analysis and management library
- **Pan**el **Da**ta **S**ystem

- Open Sourced by AQR Capital Management, LLC in late 2009

- 30.000 lines of tested Python/Cython code
- Used in production in many companies

Pandas

- Rich data structures and functions to make working with structured data fast, easy, and expressive
- Built on top of Numpy with its high performance array-computing features
- flexible data manipulation capabilities of spreadsheets and relational databases
- Sophisticated indexing functionality
  - slice, dice, perform aggregations, select subsets of data

The ideal tool for data scientists

- Cleaning data
- Analyzing data
- Modeling data
- Organizing the results of the analysis into a form suitable for plotting or tabular display

Series

- one-dimensional array-like object

```
>>> s = Series((1,2,3,4,5))
```

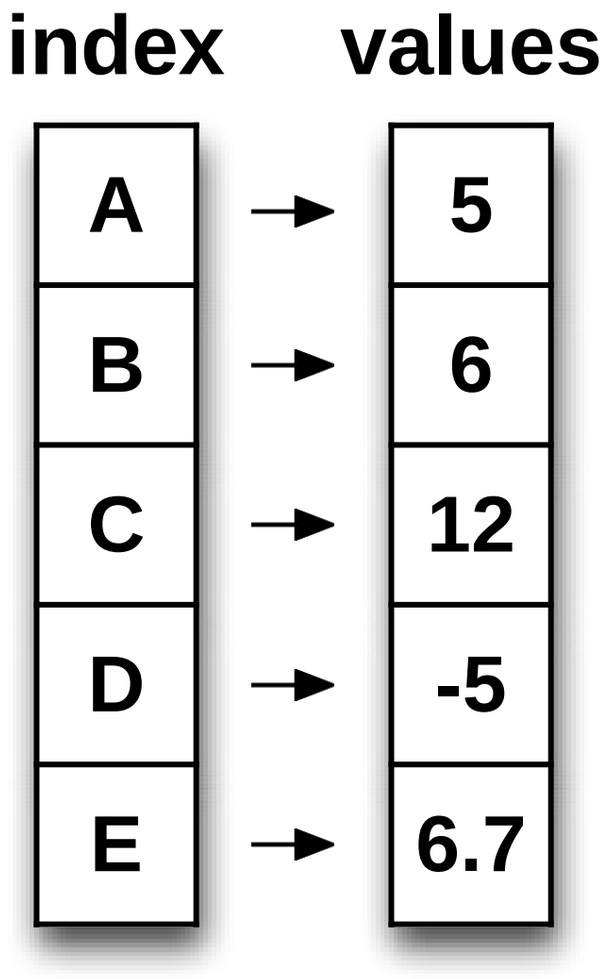
- Contains an array of data (of any Numpy data type)

```
>>> s.values
```

- Has an associated array of data labels, the *index (*Default index from 0 to N - 1)

```
>>> s.index
```

Series



- Subclass of `numpy.ndarray`
- Data: any type
- Index labels need not be ordered
- Duplicates are possible (but result in reduced functionality)

Series data structure

```
>>> import numpy
>>> randn = numpy.random.randn
>>> from pandas import *
>>> s = Series(randn(3),('a','b','c'))
>>> s
a   -0.889880
b    1.102135
c   -2.187296
>>> s.mean()
-0.65834710697853194
```

Series to/from dict

- Series to Python dict - **No more explicit order**

```
>>> dict(s)
{'a': -0.88988001423312313,
 'c': -2.1872960440695666,
 'b': 1.1021347373670938}
```

- Back to a Series with a **new Index from sorted dictionary keys**

```
>>> Series(dict(s))
a   -0.889880
b    1.102135
c   -2.187296
```

Reindexing labels

```
>>> s
a   -0.496848
b    0.607173
c   -1.570596
>>> s.index
Index([a, b, c], dtype=object)
>>> s.reindex(['c','b','a'])
c   -1.570596
b    0.607173
a   -0.496848
```

Vectorization

```
>>> s + s
a   -1.779760
b    2.204269
c   -4.374592
```

- Series work with Numpy

```
>>> numpy.exp(s)

a    0.410705
b    3.010586
c    0.112220
```

Data alignment

- Binary operations are joins!
- https://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.align.html#pandas.Series.align

B	1
C	2
D	3
E	4

+

A	0
B	1
C	2
D	3

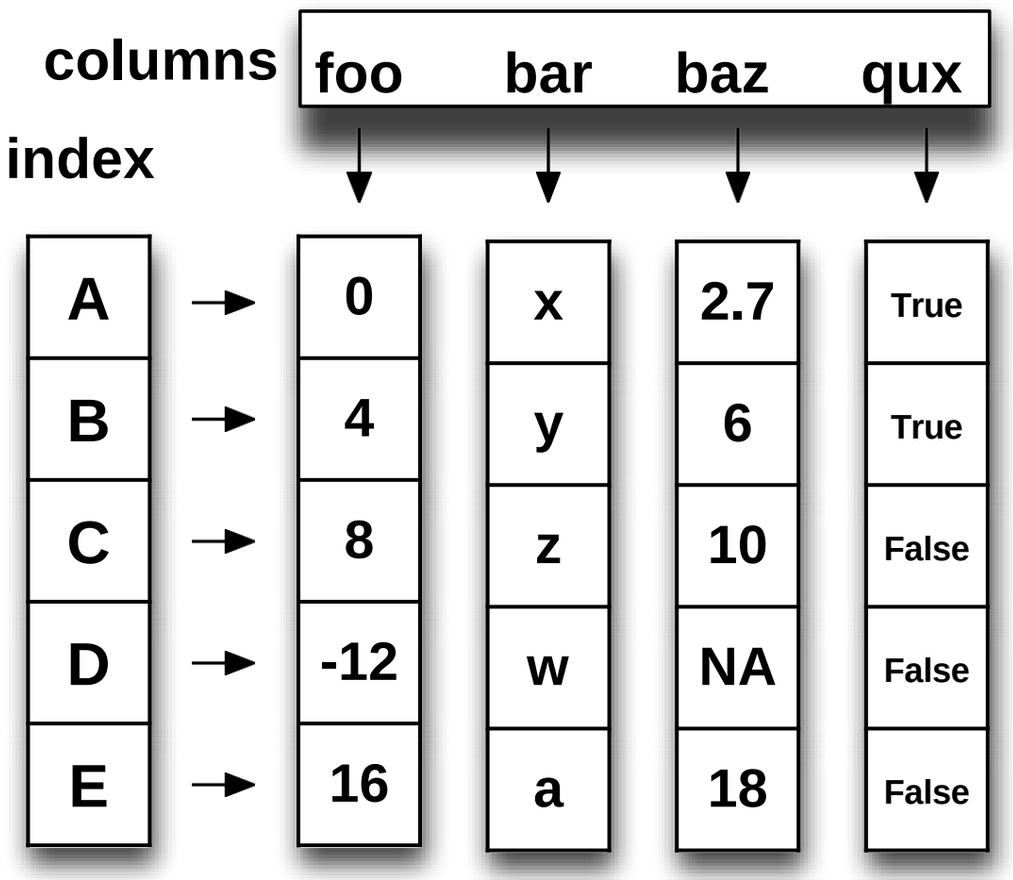
=

A	NA
B	2
C	4
D	6
E	NA

DataFrame

- Like data.frame in the statistical language/package R
- 2-dimensional tabular data structure
- Data manipulation with integrated indexing
- Support heterogeneous columns

DataFrame



- NumPy array-like
- Each column can have a different type
- Row and column index
- Size mutable: insert and delete columns

DataFrame

```
>>> d = {'one': s*s, 'two': s+s}
>>> DataFrame(d)
        one       two
a  0.791886 -1.779760
b  1.214701  2.204269
c  4.784264 -4.374592
>>> df.index
Index([a, b, c], dtype=object)
>>> df.columns
Index([one, two], dtype=objec)
```

Dataframe add column

- Add a third column

```
>>> df['three'] = s * 3
```

- It will share the existing index

```
>>> df
          one         two       three
a  0.791886 -1.779760 -2.669640
b  1.214701  2.204269  3.306404
c  4.784264 -4.374592 -6.561888
```

Access to columns

- Access by attribute

```
>>> df.one
          one
a 0.791886
b 1.214701
c 4.784264
```

- Access by dict like notation

```
>>> df['one']
          one
a 0.791886
b 1.214701
c 4.784264
```

Reindexing

```
>>> df.reindex(['c','b','a'])
>>> df
          one         two       three
c 4.784264 -4.374592 -6.561888
b 1.214701  2.204269  3.306404
a 0.791886 -1.779760 -2.669640
```

Drop entries from an axis

```
>>> df.drop('c')
b 1.214701  2.204269  3.306404
a 0.791886 -1.779760 -2.669640

>>> df.drop(['b,'a'])
        one       two     three
c 4.784264 -4.374592 -6.561888
```

Descriptive statistics

```
>>> df.mean()
one        2.263617
two       -1.316694
three     -1.975041
```

- Also: `count, sum, median, min, max, abs, prod, std, var, skew, kurt, quantile,cumsum, cumprod, cummax,cummin`

Computational Tools

- Covariance

```
>>> s1 = Series(randn(1000))
>>> s2 = Series(randn(1000))
>>> s1.cov(s2)
0.013973709323221539
```

- Also: pearson, kendall, spearman

I/O Operations

Read from
- csv, json, excel, html, SQL…

- https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html#pandas.read_csv

Data manipulations

- Join / merge *(database-like)*
- Get_dummies *(one hot vector from categorical data)*

Aggregated Statistics

- Group by
- Describe

Plotting

```
DataFrame.plot.plotType()
- hist, pie, density, box,…
```

This and much more...

- Reshaping and Pivot Tables
- Time Series / Date functionality
- Sparse data structures

Resources

- http://pypi.python.org/pypi/pandas
- http://code.google.com/p/pandas

Semestrální projekt

- Výběr ze 4-5 úkolů, případně po schválení vlastní zadání
  - Varianty budou představeny průběžně na příštích cvičeních cca do ½ dubna
  - Typicky supervised ML
  - Výběr zadání do 30.4.
- Vypracování obsahuje
  - Zdrojový kód
    - komentáře + readme file (co a jak spustit)
  - Vstupní data (pokud se liší od zadaných)
  - Popis řešení
    - Jak byla data předzpracována? (normalizace, feature engineering, další datové zdroje…)
    - Jaké algoritmy byly použity? S jakými parametry / jak byly vybrány? (vyzkoušejte alespoň 2 algoritmy s rozumným hyperparameter tuningem)
    - Přehled výsledků + rozumná vizualizace (např. vliv některých hyperparametrů na výsledek)
    - Zajímavé/netriviální části implementace? Překvapivé výsledky? Learned knowledge?
    - Očekávaný rozsah cca 1-2 stránky

Semestrální projekt – varianta 1

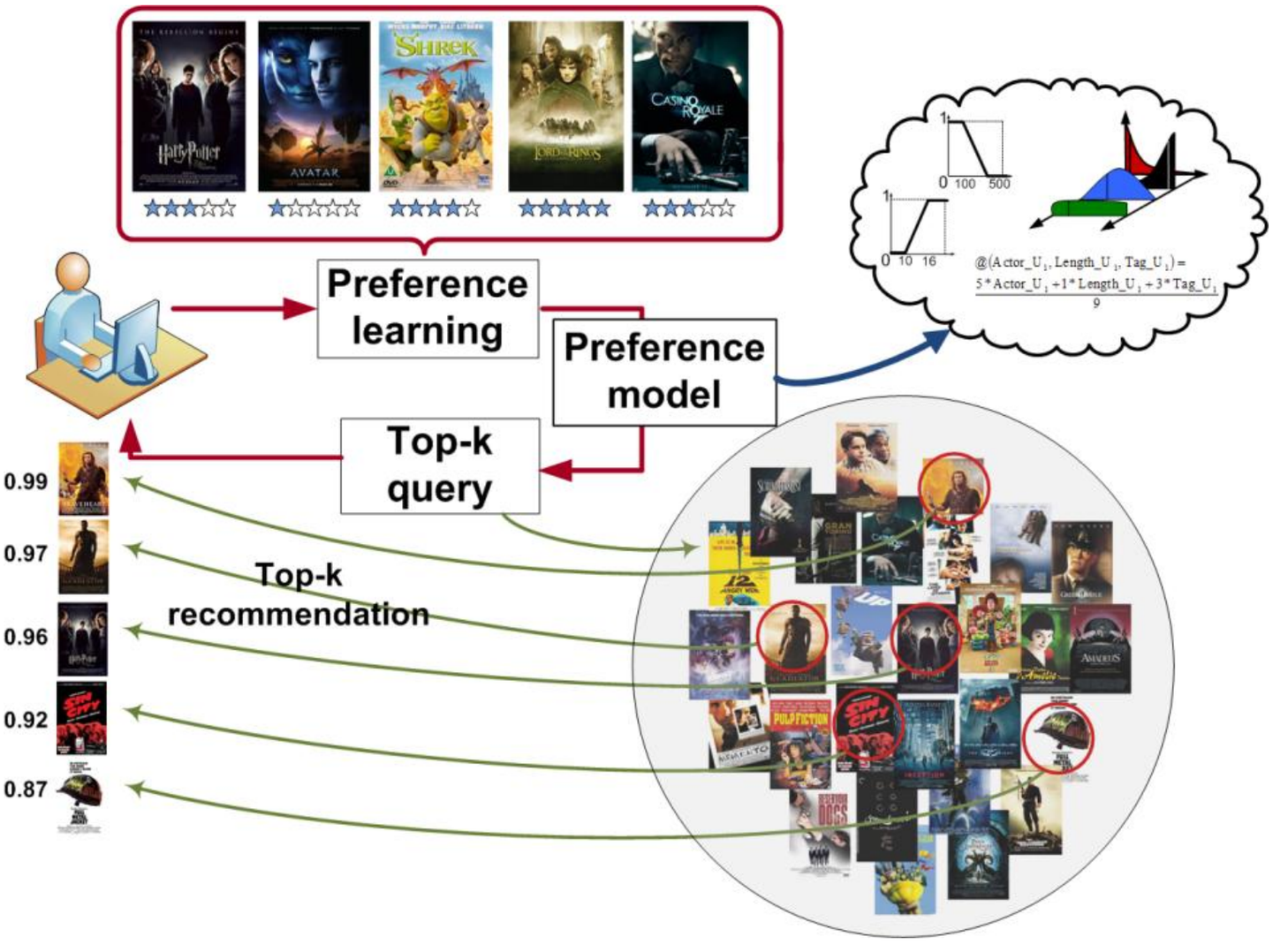
- Doporučovací systémy
  - Doporučte uživateli 10 pro něj nejzajímavějších filmů
  - MovieLens 1M dataset (cca 5000 uživatelů a filmů, 1M ratingů)
    - https://grouplens.org/datasets/movielens/
  - Známe hodnocení některých filmů uživatelem, základní data o filmech a demografická data o uživatelích
  - Možnost použít externí datové zdroje (např. IMDB)
  - Evaluace:
    - Každému uživateli bude náhodně odebráno 10% hodnocených filmů, vašim cílem je některé z nich uživateli doporučit
    - nDCG na top-10 doporučených filmech https://en.wikipedia.org/wiki/Discounted_cumulative_gain
    - Cílem je tedy ranking prediction (seřazení objektů od nejlepšího po nejhorší)
  - Odevzdané řešení musí být lepší než základní baseline:
    - Most popular (doporučuje globálně nejpopulárnější filmy)
  - Typické algoritmy:
    - KNN, nebo faktorizace matic (SVD++) na matici hodnocení uživatelů a objektů
    - Běžné ML algoritmy na vlastnostech uživatelů a objektů (vlastní model pro každého uživatele / skupinu uživatelů – clustering, nebo sjednocení vlastností uživatele a objektu)
    - Vector Space Model: uživatel (query) je definovaný jako agregace jím hodnocených objektů

Semestrální projekt – varianta 1

- Doporučovací systémy
  - Doporučte uživateli 10 pro něj nejzajímavějších filmů
  - MovieLens 1M dataset (cca 5000 uživatelů a filmů, 1M ratingů)
    - https://grouplens.org/datasets/movielens/
  - Známe hodnocení některých filmů uživatelem, základní data o filmech a demografická data o uživatelích
  - Možnost použít externí datové zdroje (např. IMDB)
  - Evaluace:
    - Každému uživateli bude náhodně odebráno 10% hodnocených filmů, vašim cílem je některé z nich uživateli doporučit
    - nDCG na top-10 doporučených filmech https://en.wikipedia.org/wiki/Discounted_cumulative_gain
    - Cílem je tedy ranking prediction (seřazení objektů od nejlepšího po nejhorší)
  - Odevzdané řešení musí být lepší než základní baseline:
    - Most popular (doporučuje globálně nejpopulárnější filmy)
  - Typické algoritmy:
    - KNN, nebo faktorizace matic (SVD++) na matici hodnocení uživatelů a objektů
    - Běžné ML algoritmy na vlastnostech uživatelů a objektů (vlastní model pro každého uživatele / skupinu uživatelů - clustering)
    - Vector Space Model: uživatel (query) je definovaný jako agregace jím hodnocených objektů
    - http://www.ksi.mff.cuni.cz/~peska/vyuka/nswi166/

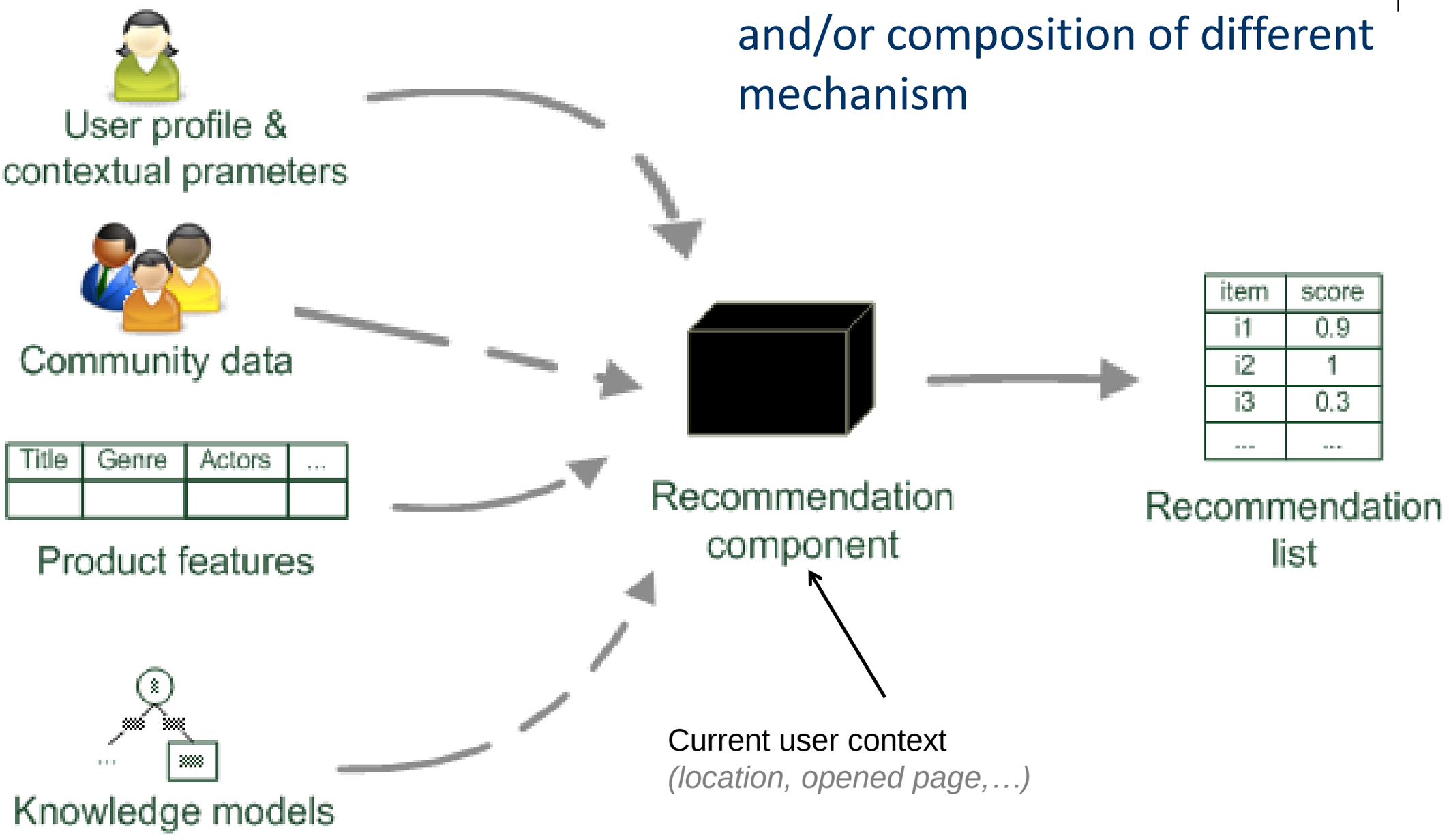
Recommender Systems
Basic Workflow

- ***User*** provides ***feedback*** from which user preferences are learned
- Recommendations are provided based on
  - **User feedback**
  - Object attributes
  - User attributes/relations
  - Context
  - …



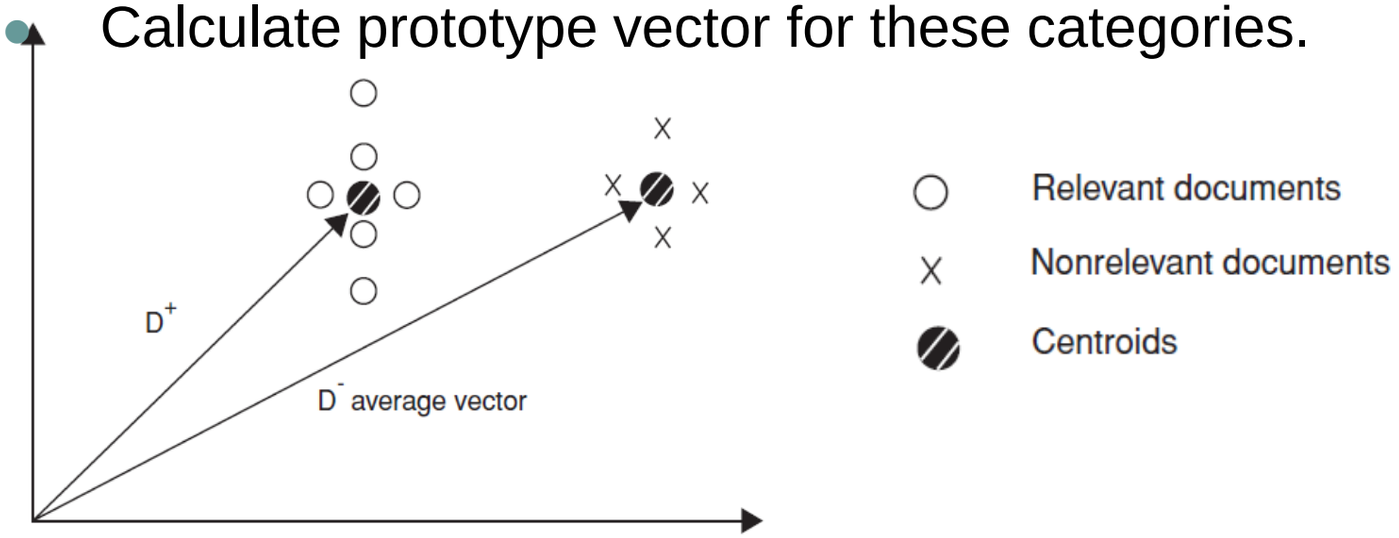
Paradigms of recommender systems

Hybrid: combinations of various inputs and/or composition of different mechanism



Vector Space Model

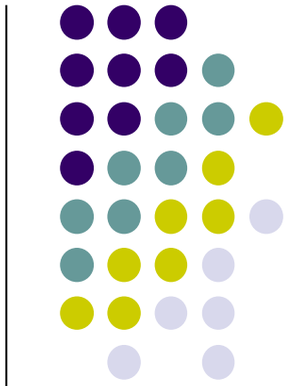
- Document collections D^+ (liked) and D^- (disliked)
  - Calculate prototype vector for these categories.



- Computing modified query Q_{i+1} from current query Q_i with:

$$Q_{i+1} = \alpha * Q_i + \beta\left(\frac{1}{|D^+|}\sum_{d^+ \in D^+} d^+\right) - \gamma\left(\frac{1}{|D^-|}\sum_{d^- \in D^-} d^-\right)$$

  - α, β, γ used to fine-tune the feedback
    - α weight for original query
    - β weight for positive feedback
    - γ weight for negative feedback
  - Often only positive feedback is used
    - More valuable than negative feedback



Matrix Completion (Matrix factorization)

Matrix completion

- Given a sparse matrix
- We want to fill-in the
- unknown values
- The values of the matrix
- are dependent on
- each other

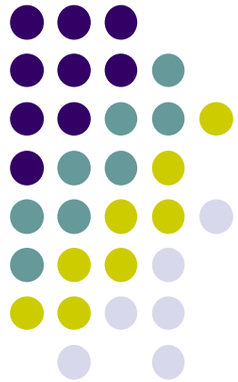
5	?	1	?	?	…
?	?	5	?	4	…
5	4	2	?	?	…
?	3	?	2	5	…
1	?	5	?	4	…
5	4	?	?	2	…
…	…	…	…	…	…

- Approaches
  - Search for similar rows/columns
  - (nearest neighbour collaborative filtering)
  - Matrix factorization
  - Restricted Boltzmann Machines (RBM)
  - …
  -

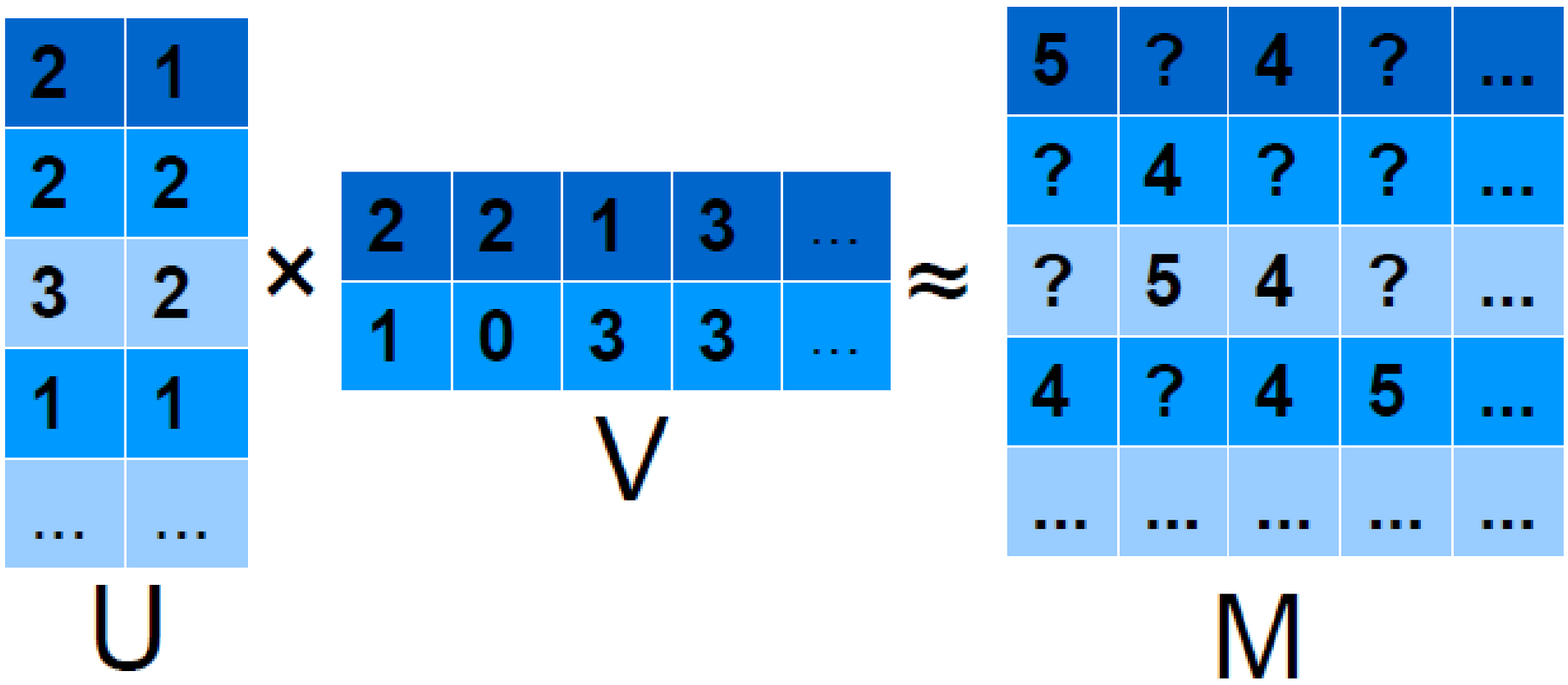
Example: Nearest neighbor collaborative filtering for movie-rating prediction (recommender systems)

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	
User 1	5	?	1	?	?	…
User 2	?	?	5	?	4	…
User 3	5	4	2	?	?	…
User 4	?	3	?	2	5	…
User 5	1	?	5	?	4	…
User 6	5	4	?	?	2	…
	…	…	…	…	…	…

Matrix factorization



- We estimate matrix *M* as the product of two matrices *U* and *V*.
- Based on the known values of *M*, we search for *U* and *V* so that their product best estimates the (known) values of *M*



Problem formulation

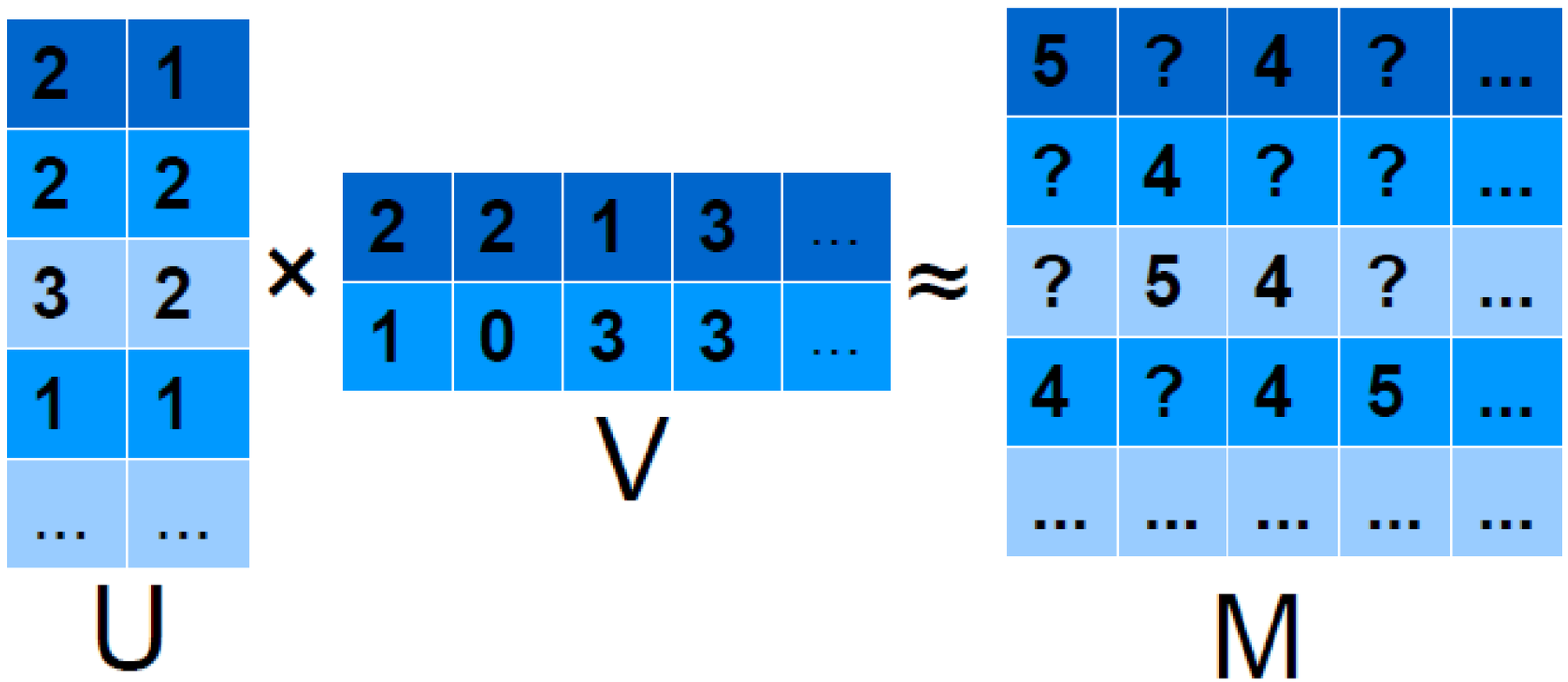
- Target function:
- sum of squared errors + regularization
-

$$\sum_{i,j} \left(m_{i,j} - \sum_{k=0}^{K} u_{i,k} v_{k,j}\right)^2 + \lambda\left(\sum_{i,j} u_{i,j}^2 + \sum_{i,j} v_{i,j}^2\right)$$

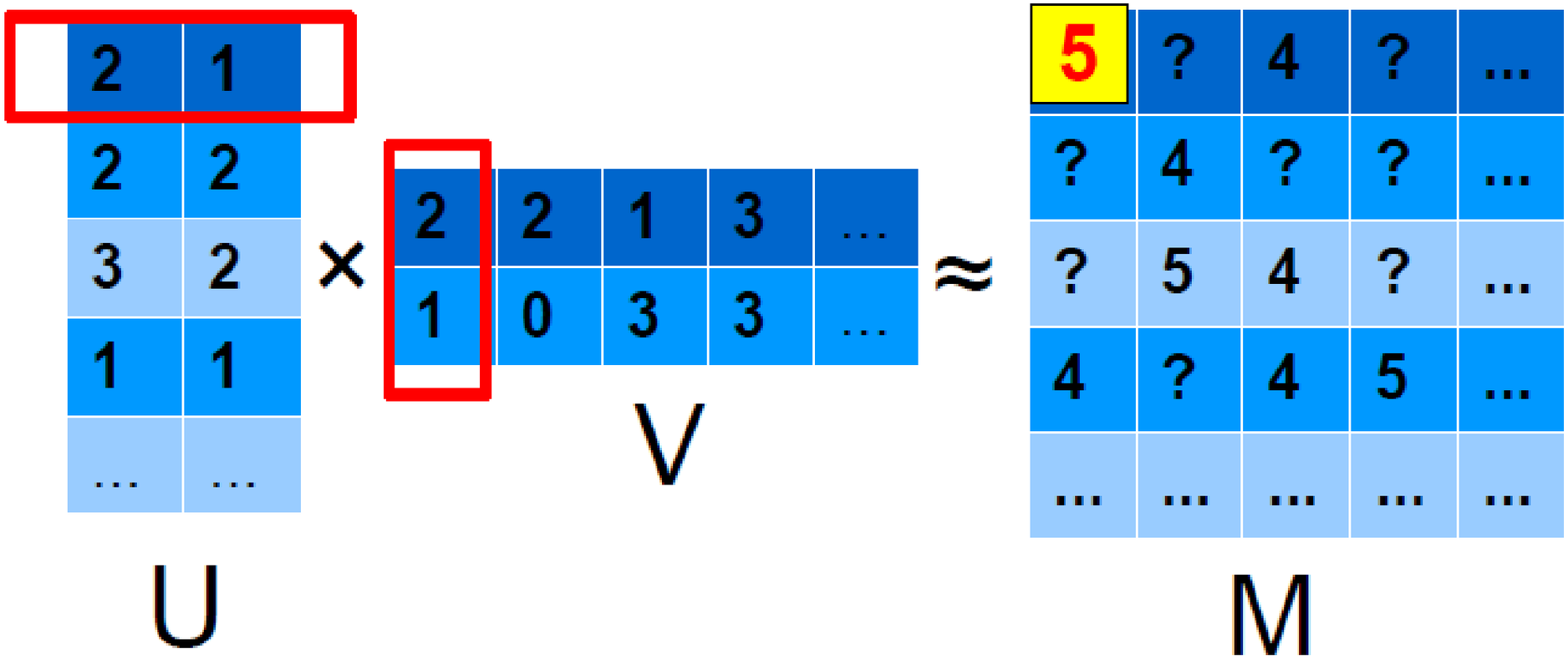
-
- where λ is the weight of the regularization term
- (i. e., a constant giving the importance of the
- regularization term)
- Minimization of the above loss function using stochastic gradient descent (or any other optimization algorithms)

High-level view of matrix factorization algorithm

- Random initialization of *U* and *V*
- While *U* x *V* does not approximate the known values
- of *M* well enough
  - Choose a known value of *M,* we denote it by *x*
  - Adjust the values of the corresponding row and column of U and V respectively, so that the approximation becomes better



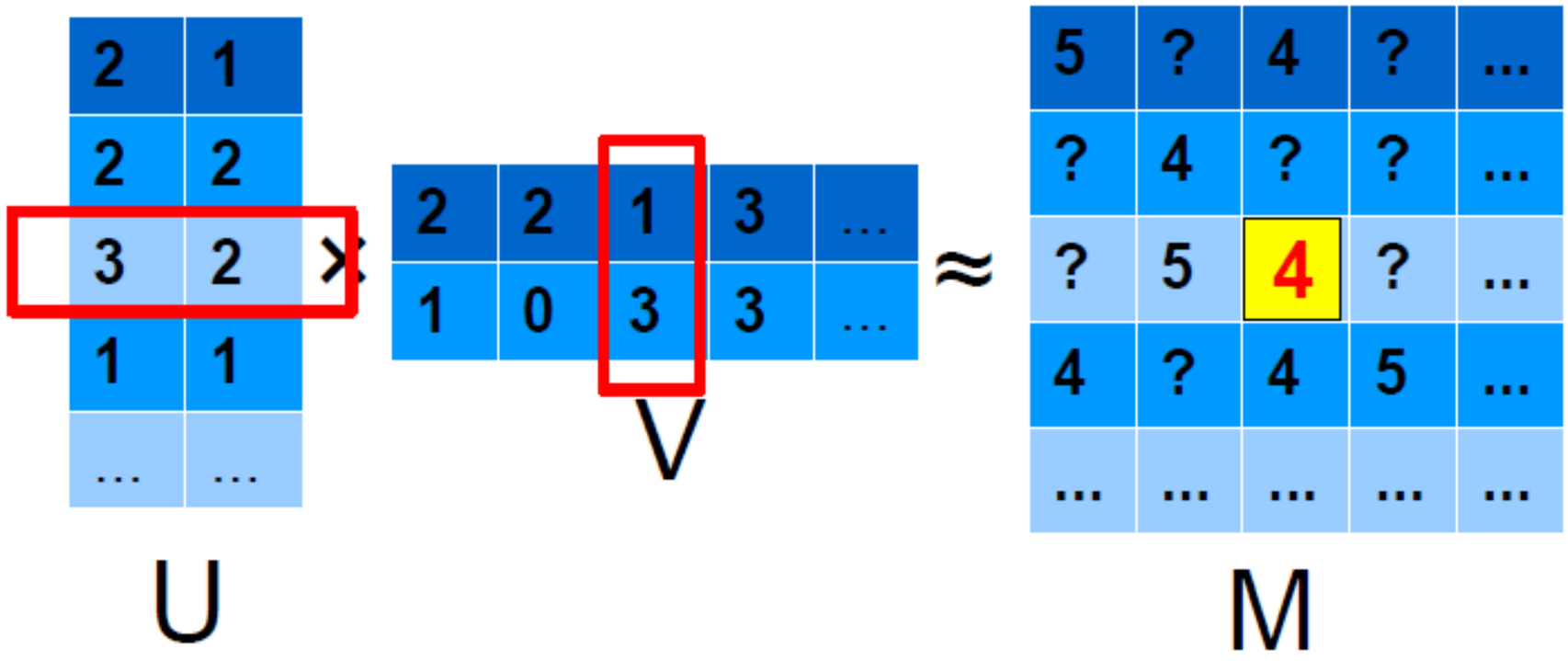
Example for an adjustment step



(2*2)+(1*1) = 5 which equals to the selected value

→ we do not do anything

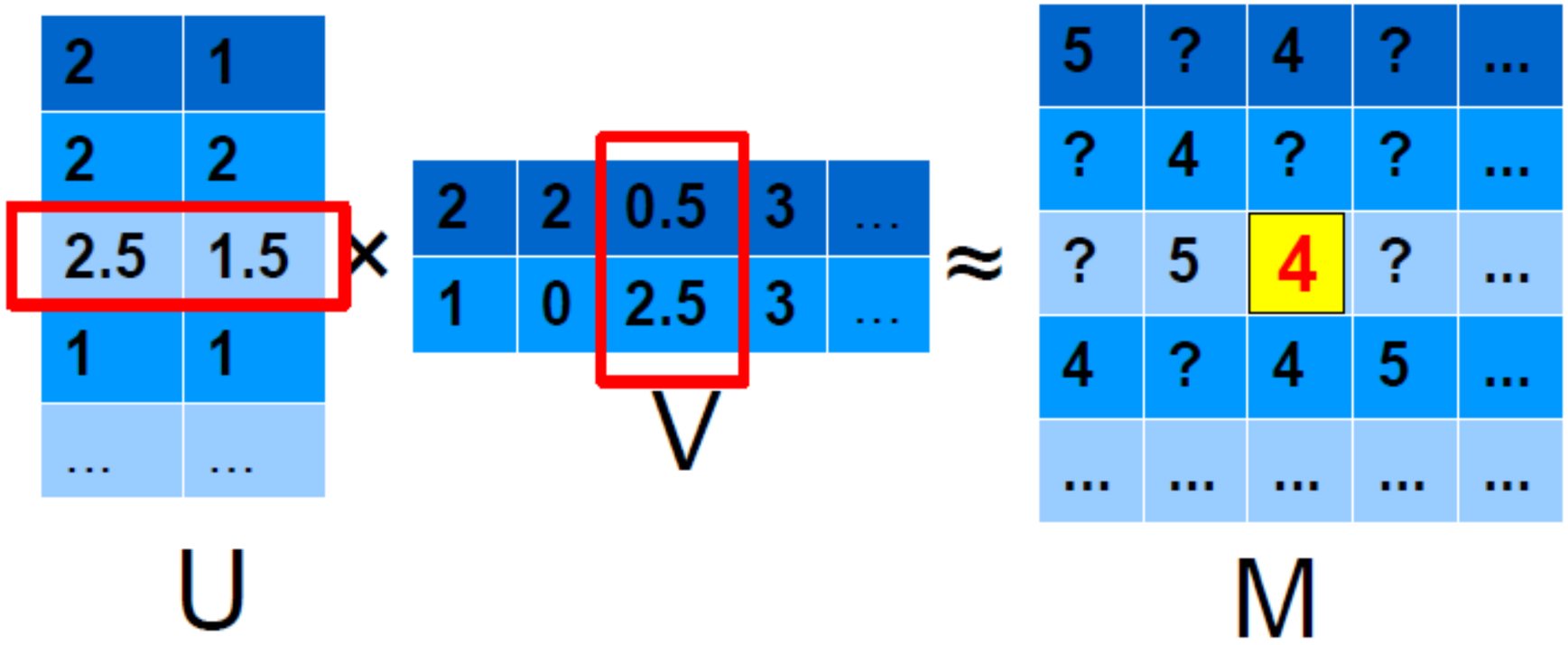
Example for an adjustment step



(3*1)+(2*3) = 9

9 > 4 → we decrease the values of the corresponding rows so that their products will be closer to 4

Example for an adjustment step

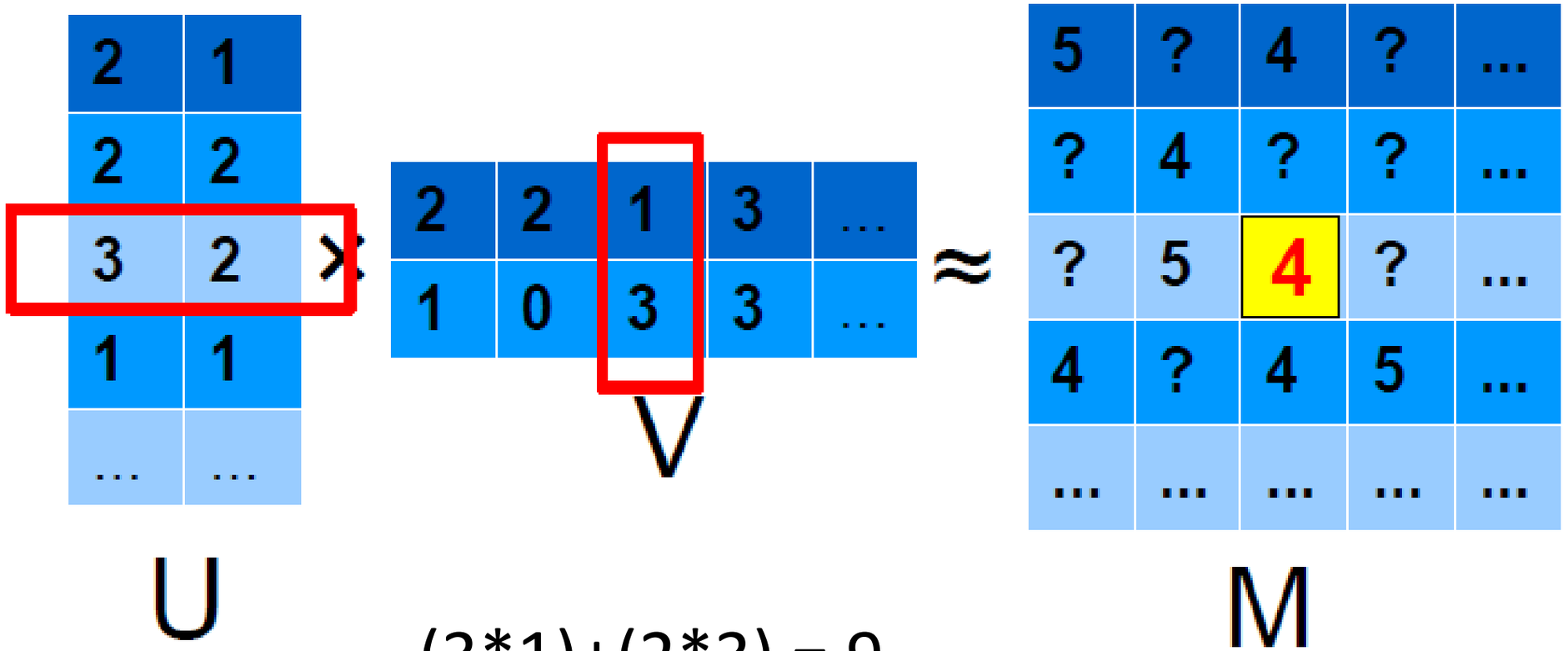


(3*1)+(2*3) = 9

9 > 4 → we decrease the values of the corresponding rows so that their products will be closer to 4

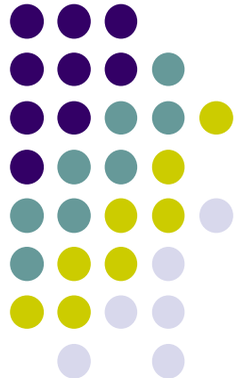
Why is the algorithm „good”?

- 1. The adjustment should be proportional to the error → let it be ε-times the error
  - In the current example: error = 9 – 4 = 5
  - with ε=0.1 we will decrease all the values in the corresponding rows and columns by 0.1*5=0.5

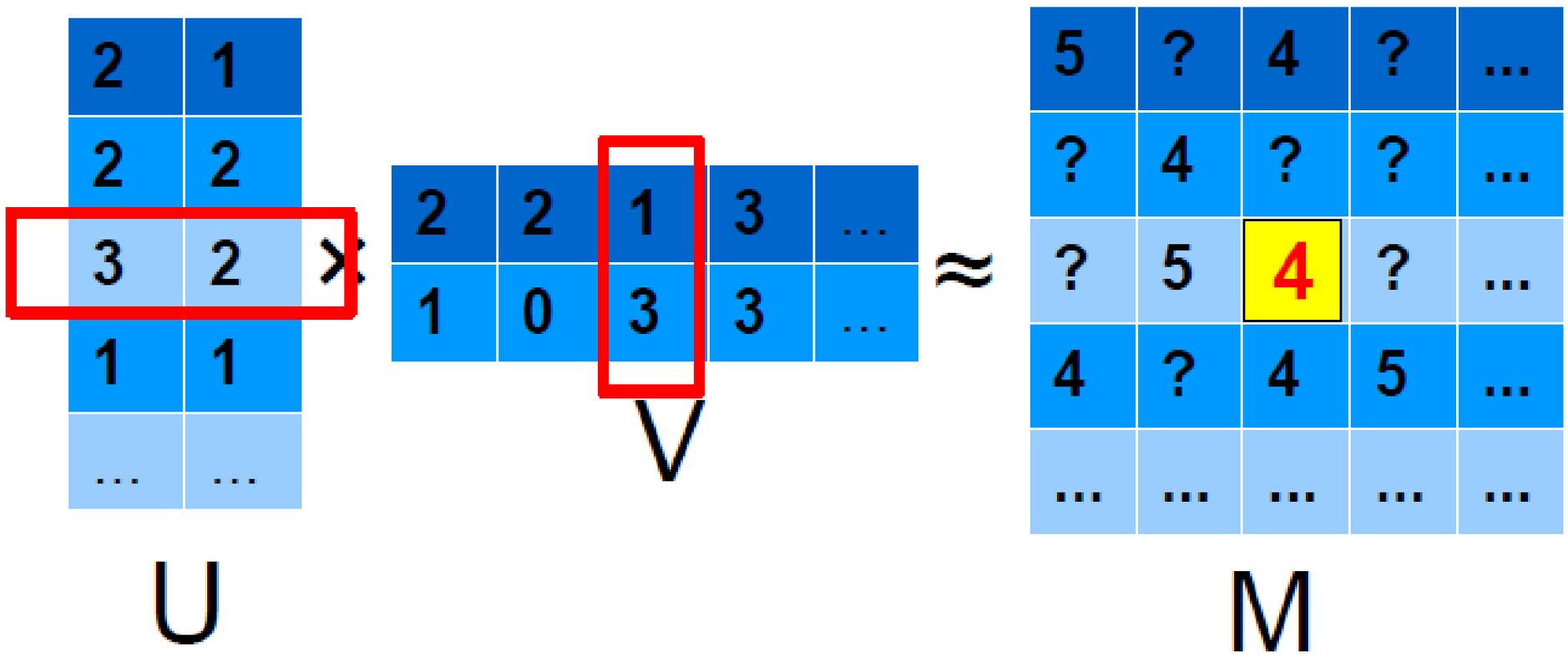


(3*1)+(2*3) = 9

Why is the algorithm „good”?



- 2. We should take into account how much each value of the row/column contributes to the error
  - For the selected row:
  - 3 is multiplied by 1 → 3 is adjusted by ε*5*1 = 0.5
  - 2 is multiplied by 3 → 2 is adjusted by ε*5*3 = 1.5
  - For the selected column respectively:
  - ε*5*3=1.5 and ε*5*2=1.0



Why is the algorithm „good”?

- 3. We prefer simpler models (avoid overfitting).
- At each adjustment step: subtract additionally
- λ-times the value
  - For the selected row: subtract additionally
  - λ*3 from 3, and λ*2 from 2 .
  - For the selected column respectively: λ*1 and λ*3
  -