Machine Learning for Large-Scale Data Analysis and Decision Making (MATH80629A) Fall 2021

Week #13- **Summary**

Announcement

- **Last Quiz, Quiz#6 on RL:** From week 13 and week 14
- **Project Presentation: in-person** on **December 6**
- **Project Report:** due **December 20**

Today

- Recommender Systems: Matrix Factorization
- Summary of Sequential decision making I
- Q&A
- Hands on session

Matrix-Factorization

Slides are collected from:
Matt Gormley at CMU
Markus Freitag, Jan-Felix Schwarz from University Potsdam
Jure Leskovec from Stanford

Types of Recommender Systems

Content Filtering

- *Example*: Pandora.com music recommendations (Music Genome Project)
- **Con:** Assumes access to side information about items (e.g. properties of a song)
- **Pro:** Got a new item to add? No problem, just be sure to include the side information

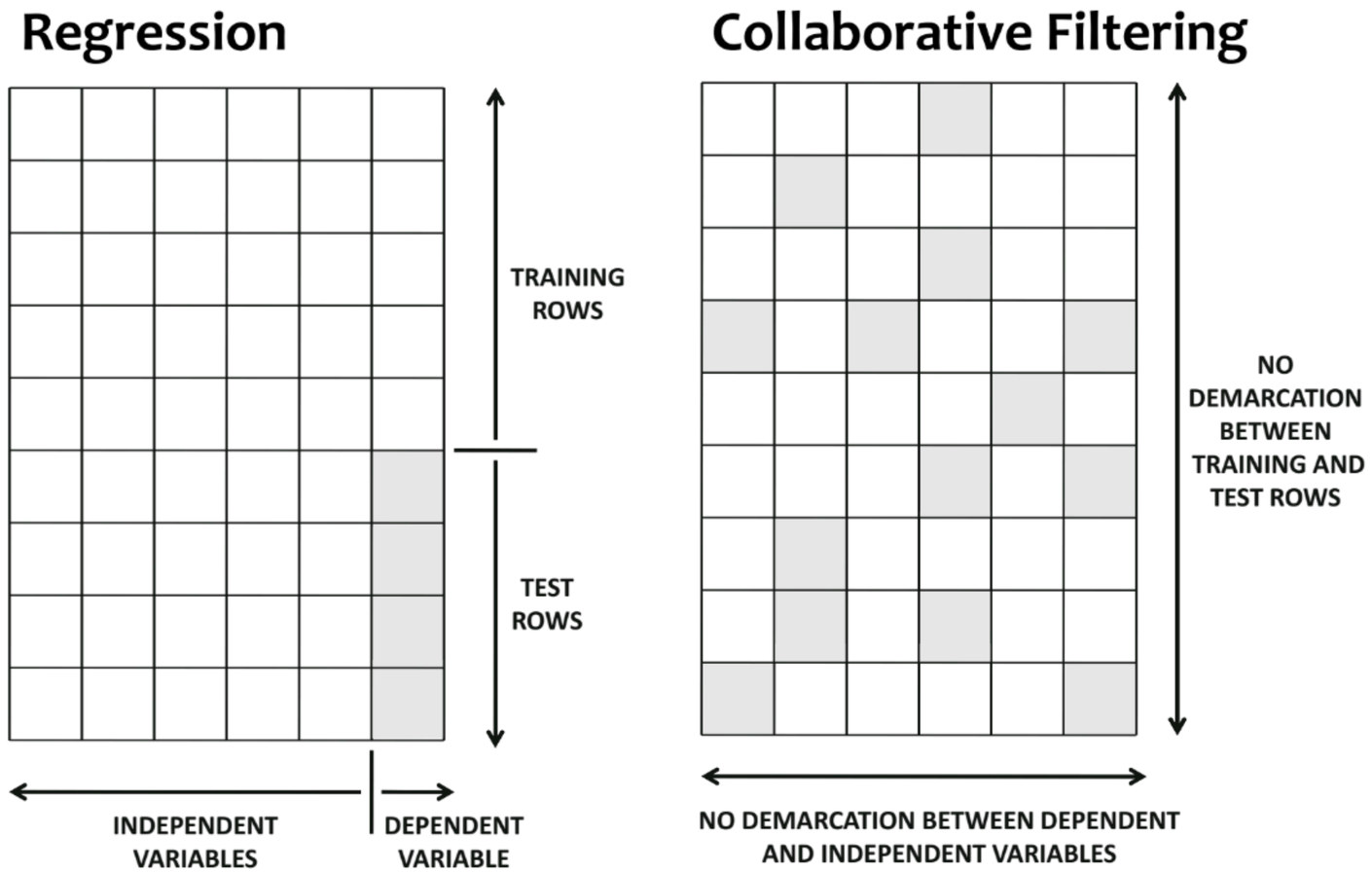
Collaborative Filtering

- *Example*: Netflix movie recommendations
- **Pro:** Does not assume access to side information about items (e.g. does not need to know about movie genres)
- **Con:** Does not work on new items that have no ratings

Collaborative Filtering

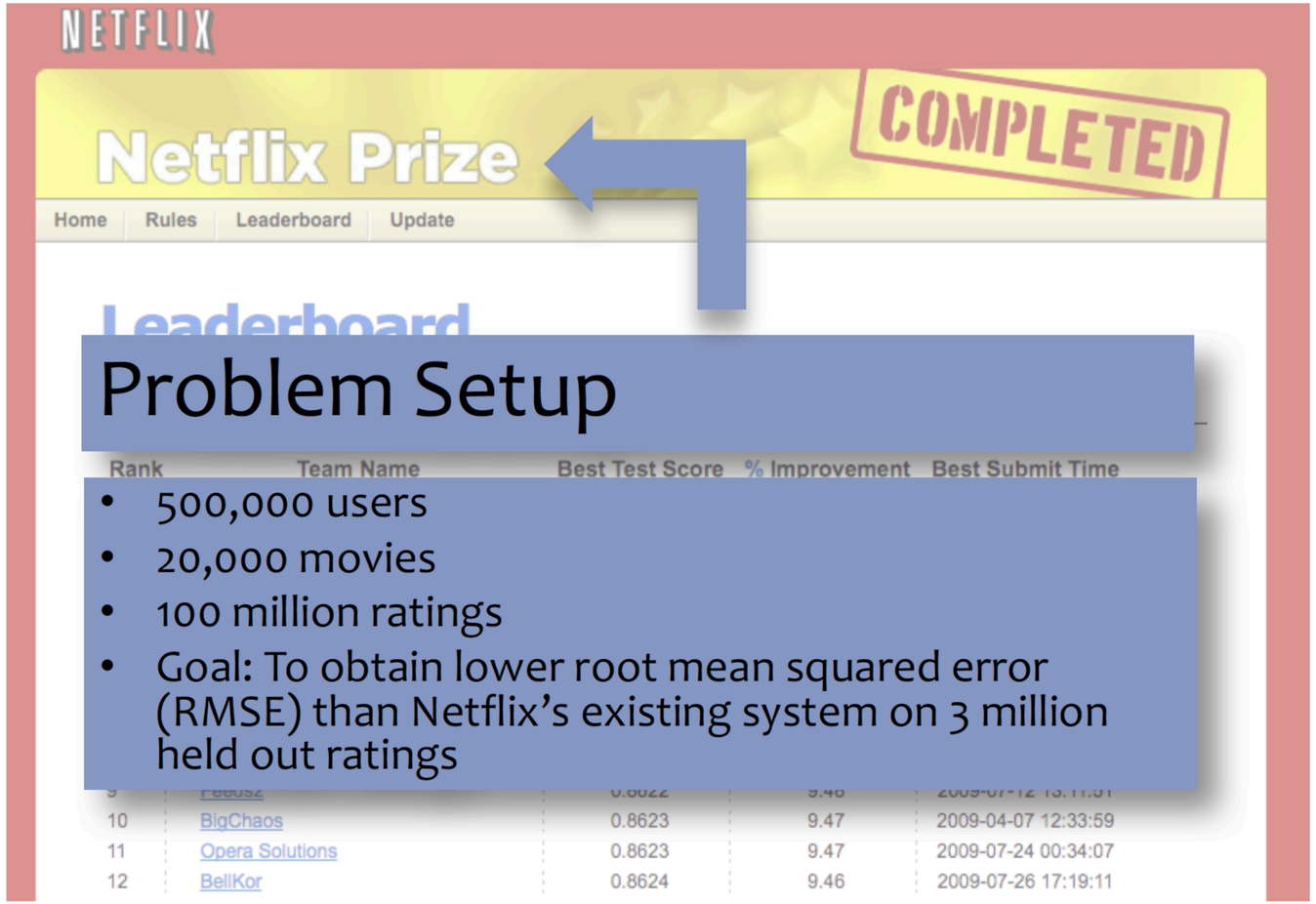
- **Everyday Examples of Collaborative Filtering…**
  - Bestseller lists
  - Top 40 music lists
  - The "recent returns" shelf at the library
  - Unmarked but well-used paths thru the woods
  - The printer room at work
  - "Read any good books lately?"
  - …
- **Common insight:** personal tastes are correlated
  - If Alice and Bob both like X and Alice likes Y then Bob is more likely to like Y
  - especially (perhaps) if Bob knows Alice

Regression vs. Collaborative Filtering

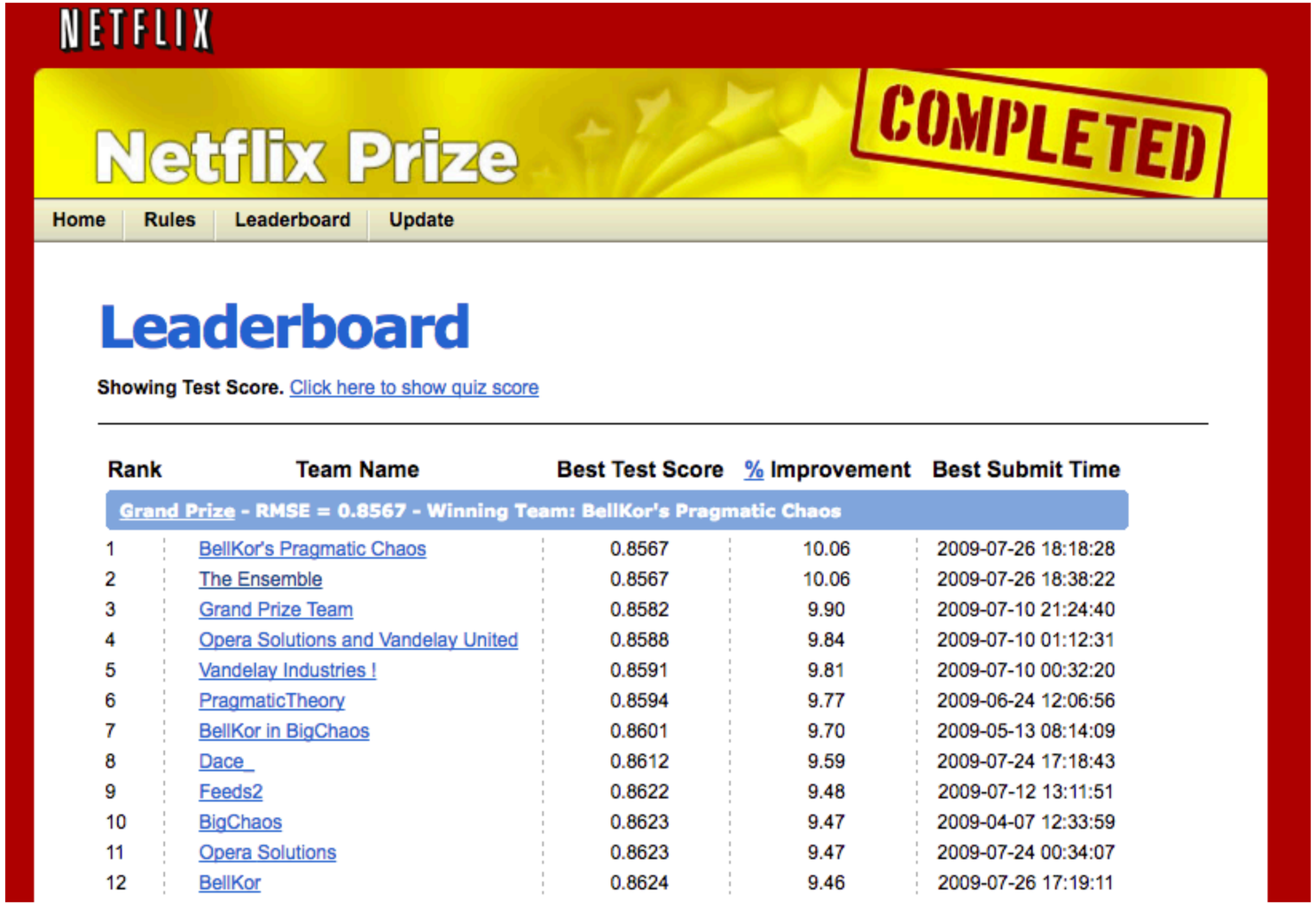


Netflix Prize

Problem Setup



Leaderboard



Matrix Factorization

MATRIX FACTORIZATION TECHNIQUES FOR RECOMMENDER SYSTEMS

Yehuda Koren, *Yahoo Research*
Robert Bell and Chris Volinsky, *AT&T Labs—Research*

As the Netflix Prize competition has demonstrated, matrix factorization models are superior to classic nearest-neighbor techniques for producing product recommendations, allowing the incorporation of additional information such as implicit feedback, temporal effects, and confidence levels.

Modern consumers are inundated with choices. Electronic retailers and content providers offer a huge selection of products, with unprecedented opportunities to meet a variety of special needs and tastes. Matching consumers with the most appropriate products is key to enhancing user satisfaction and loyalty. Therefore, more retailers have become interested in recommender systems, which analyze patterns of user interest in products to provide personalized recommendations that suit a user's taste. Because good personalized recommendations can add another dimension to the user experience, e-commerce leaders like Amazon.com and Netflix have made recommender systems a salient part of their websites.

Such systems are particularly useful for entertainment products such as movies, music, and TV shows. Many customers will view the same movie, and each customer is likely to view numerous different movies. Customers have proven willing to indicate their level of satisfaction with particular movies, so a huge volume of data is available about which movies appeal to which customers. Companies can analyze this data to recommend movies to particular customers.

RECOMMENDER SYSTEM STRATEGIES

Broadly speaking, recommender systems are based on one of two strategies. The *content filtering* approach creates a profile for each user or product to characterize its nature. For example, a movie profile could include attributes regarding its genre, the participating actors, its box office popularity, and so forth. User profiles might include demographic information or answers provided on a suitable questionnaire. The profiles allow programs to associate users with matching products. Of course, content-based strategies require gathering external information that might not be available or easy to collect.

A known successful realization of content filtering is the Music Genome Project, which is used for the Internet radio service Pandora.com. A trained music analyst scores

 Published by the IEEE Computer Society 0018-9162/09/$26.00 © 2009 IEEE

Yehuda Koren, Yahoo Research

Robert Bell and Chris Volinsky, AT&T Labs-Research

Paper published in August 2009

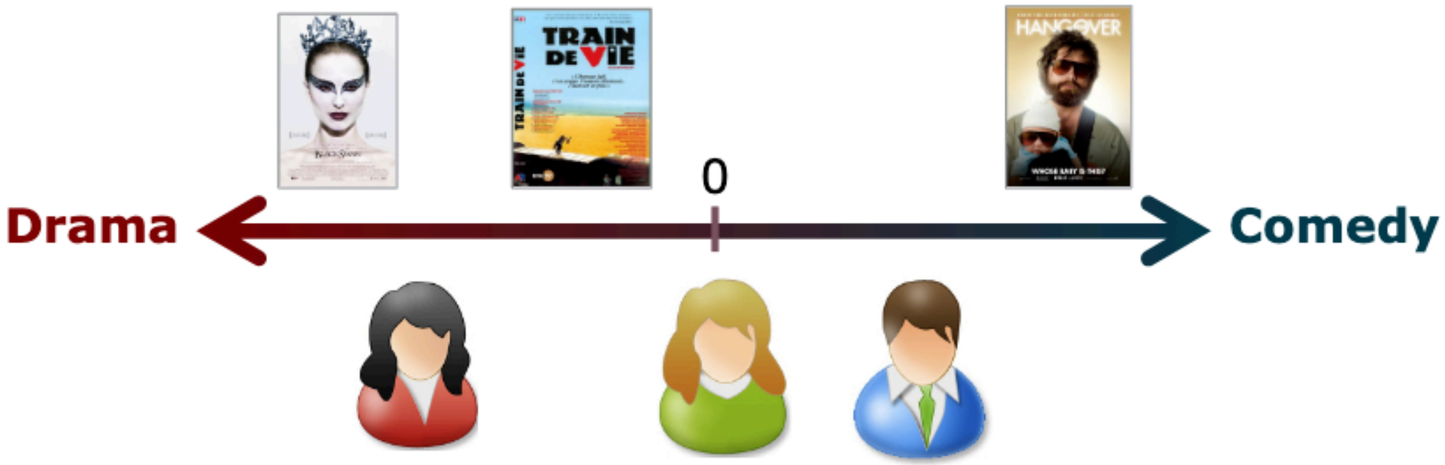
Authors won the grand Netflix Prize in September 2009

Latent Factor Models

Find features that describe the characteristics of rated objects

Item characteristics and user preferences are described with numerical factor values

Assumption: Ratings can be inferred from a model put together from a smaller number of parameters



Latent Factor Models

Items and users are associated with a factor vector

Dot product captures the user's estimated interest in the item:

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

Challenge: How to compute a mapping of items and users to factor vectors?

Approaches:

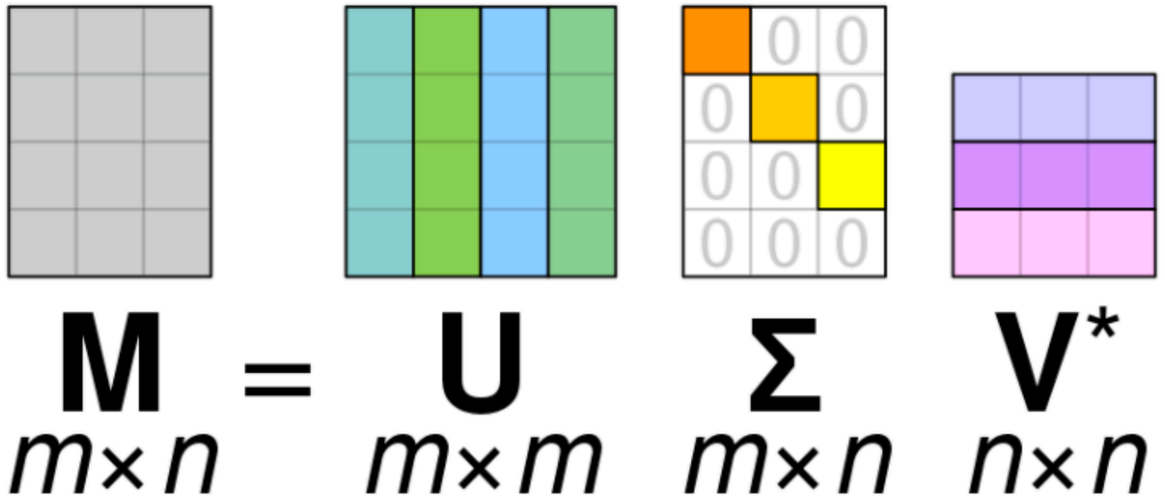
- Singular Value Decomposition (SVD)
- Matrix Factorization

Singular Value Decomposition (SVD)

- Method from **linear algebra** that has been generally used as a **dimensionality reduction** technique in machine learning.

- SVD reduces the space dimension from N-dimension to F-dimension (where F<N).

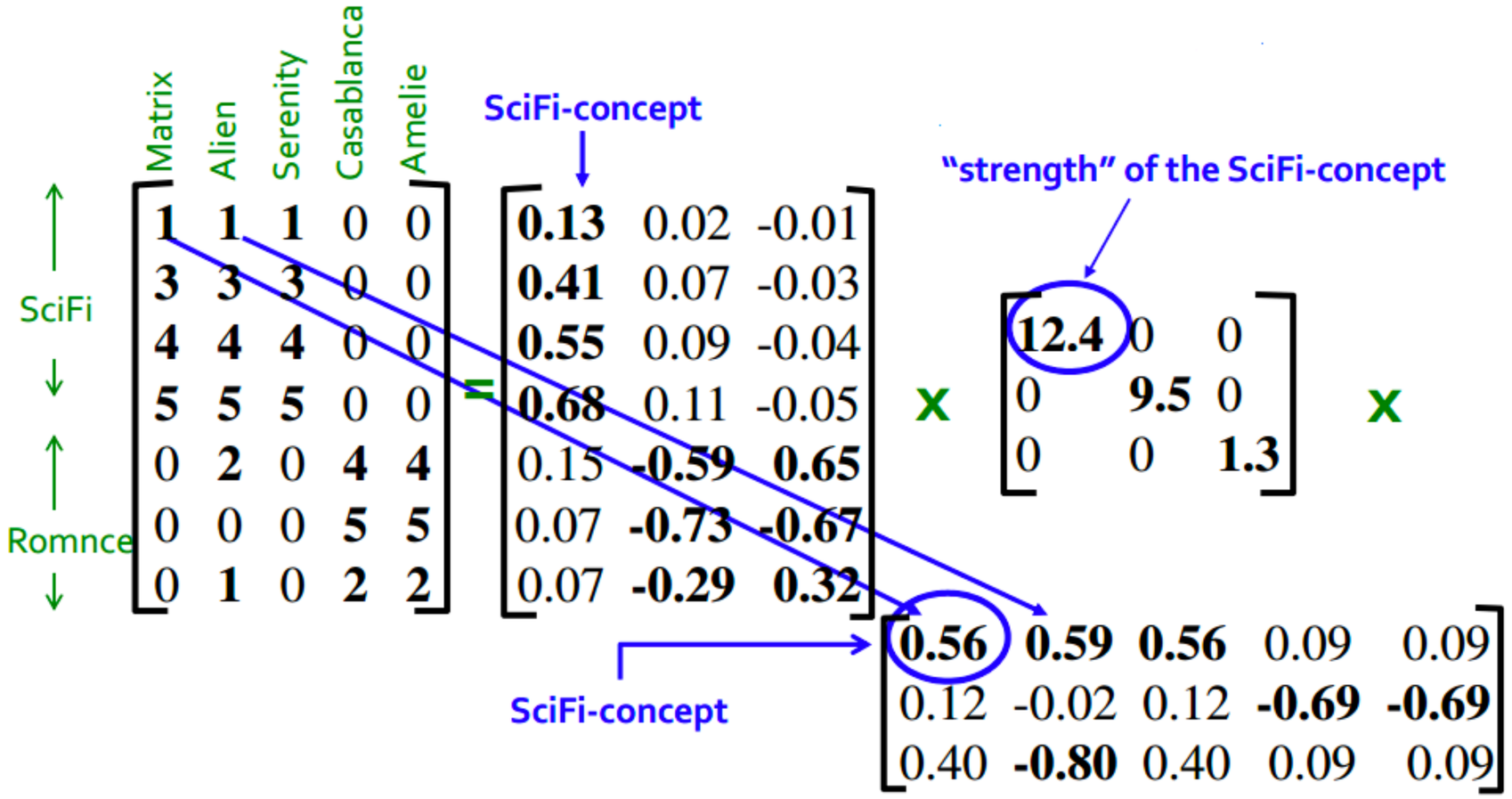
SVD



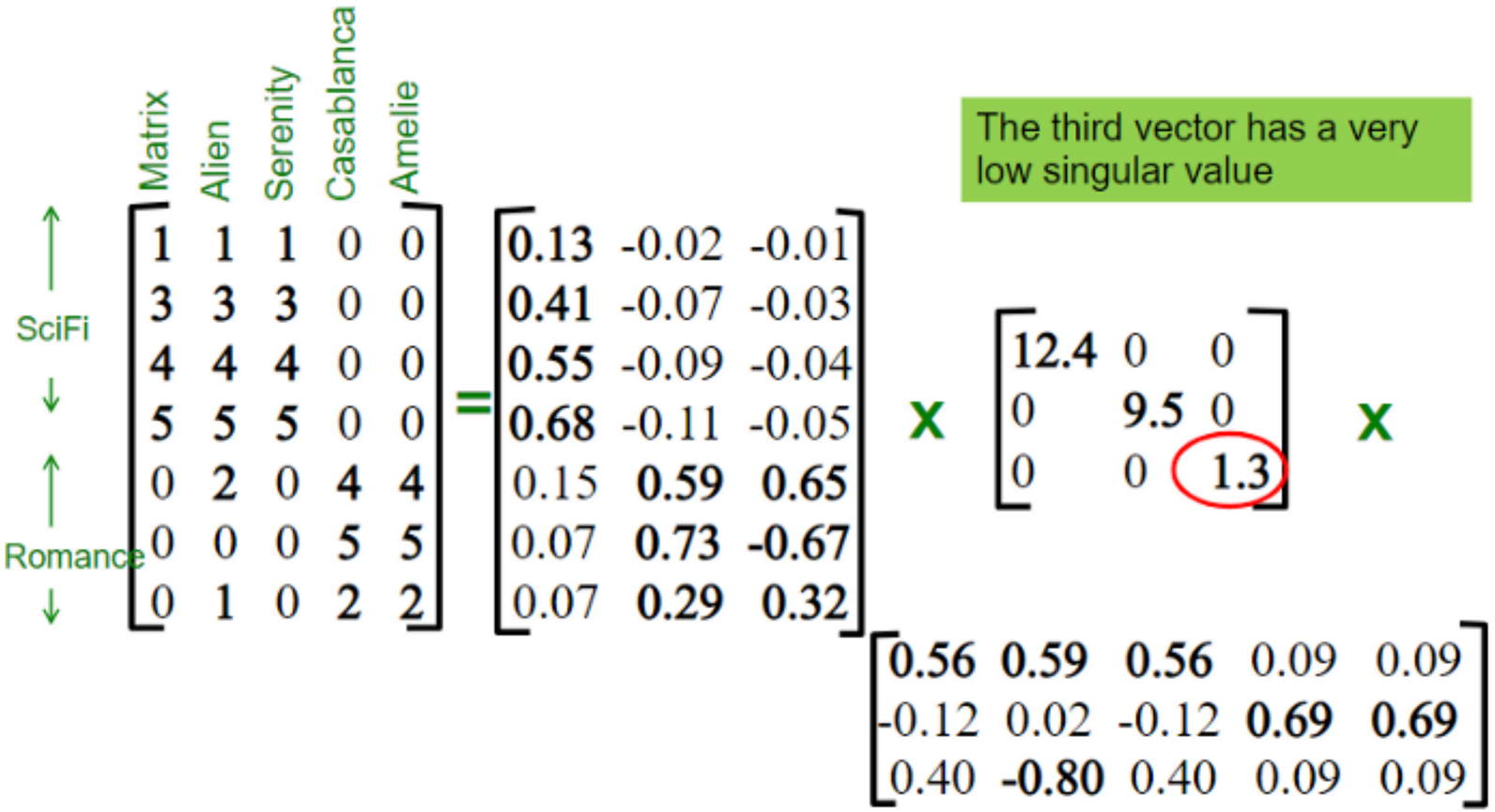
Dimensionality Reduction with SVD | Stanford University

https://youtu.be/UyAfmAZU_WI

Example


Matrix
Alien
Serenity
Casablanca
Amelie
SciFi
Romnce
SciFi-concept
"strength" of the SciFi-concept
SciFi-concept
1 1 1 0 0
3 3 3 0 0
4 4 4 0 0
5 5 5 0 0
0 2 0 4 4
0 0 0 5 5
0 1 0 2 2
0.13 0.02 -0.01
0.41 0.07 -0.03
0.55 0.09 -0.04
0.68 0.11 -0.05
0.15 -0.59 0.65
0.07 -0.73 -0.67
0.07 -0.29 0.32
12.4 0 0
0 9.5 0
0 0 1.3
0.56 0.59 0.56 0.09 0.09
0.12 -0.02 0.12 -0.69 -0.69
0.40 -0.80 0.40 0.09 0.09

Dimensionality reduction



Dimensionality reduction

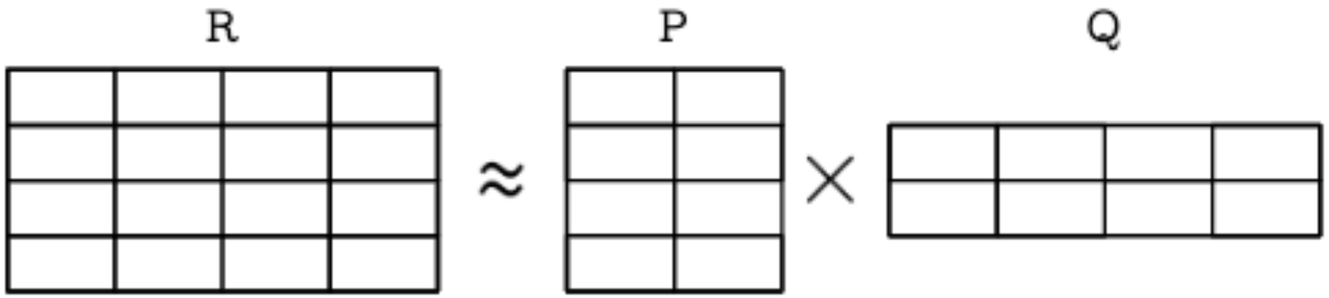
$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix} \approx \begin{bmatrix} \mathbf{0.13} & -0.02 \\ \mathbf{0.41} & -0.07 \\ \mathbf{0.55} & -0.09 \\ \mathbf{0.68} & -0.11 \\ 0.15 & \mathbf{0.59} \\ 0.07 & \mathbf{0.73} \\ 0.07 & \mathbf{0.29} \end{bmatrix} \times \begin{bmatrix} 12.4 & 0 \\ 0 & 9.5 \end{bmatrix} \times \begin{bmatrix} \mathbf{0.56} & \mathbf{0.59} & \mathbf{0.56} & 0.09 & 0.09 \\ -0.12 & 0.02 & -0.12 & \mathbf{0.69} & \mathbf{0.69} \end{bmatrix}$$

Issues of using SVD

- Calculating the SVD consists of finding the eigenvalues and eigenvectors of rating matrix and it's transpose
- *Computationally expensive for large data (as it is linear in the size of the data)*
- *SVD assumes c*omplete input matrix: all entries available and considered. Conventional SVD is undefined for incomplete matrices!
- Large portion of missing values
- Heuristics to pre-fill missing values
    - item's average rating
    - missing values as zeros

Matrix Factorization

the completion is driven by a factorization

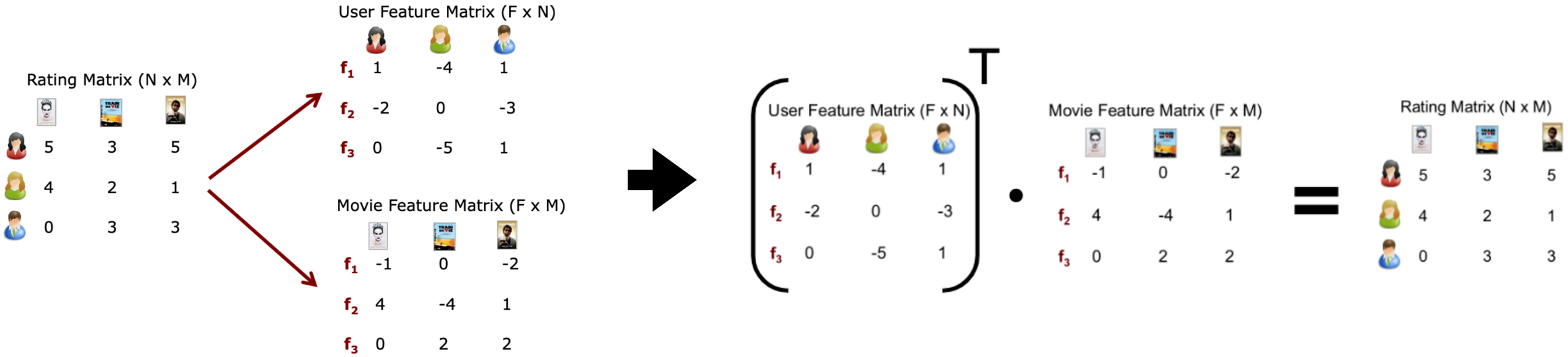


associate a latent factor vector with each user and each item

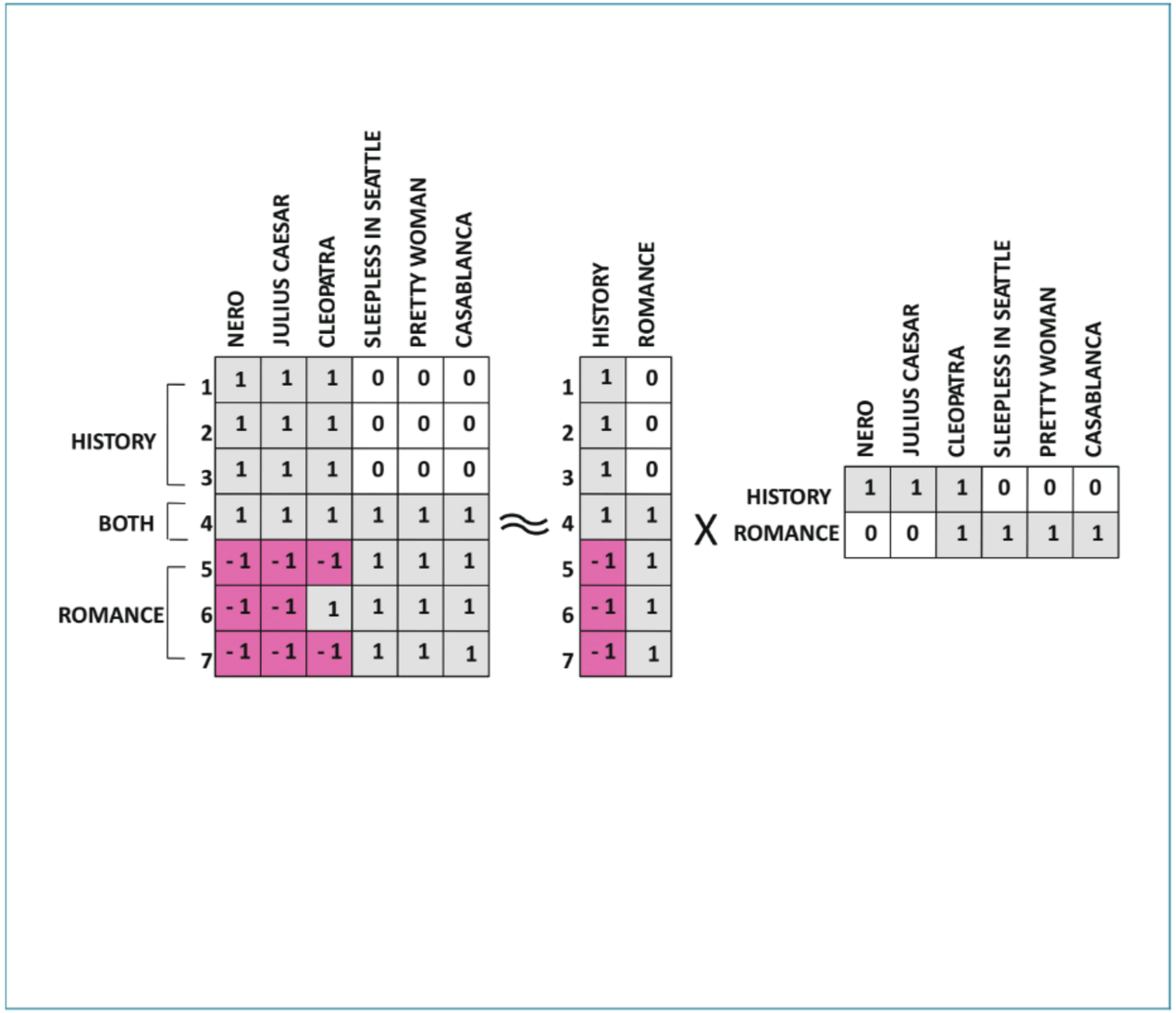
missing entries are estimated through the dot product

$$r_{ij} \approx p_i q_j$$

Matrix Factorization



Example of MF for Netflix problem



Matrix Factorization

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

r_{ui} : known rating of user u for item i

remember:
predicted rating $\hat{r}_{ui} = q_i^T p_u$

Regularization to avoid overfitting

Idea: penalize complexity

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

known as Funk SVD

λ : constant to control the extend of regularization

→ determined by cross-validation

Learning algorithms

Stochastic gradient descent

- Modification of parameters (q_i, p_u) relative to prediction error
- Recommended algorithm

Alternating least squares

- Allows massive parallelization
- Better for densely filled matrices

Sequential decision making I

Reinforcement Learning

- Sometimes we need a model where **the learning** and **the decision making** interact closely
- Imagine building a robot that must navigate autonomously
  - The robot has wheels and a camera
- You think about using a **two-stage approach:**
  1. Use supervised learning to identify objects in scenes
  2. Given scene content have a decision-making module that controls its wheels

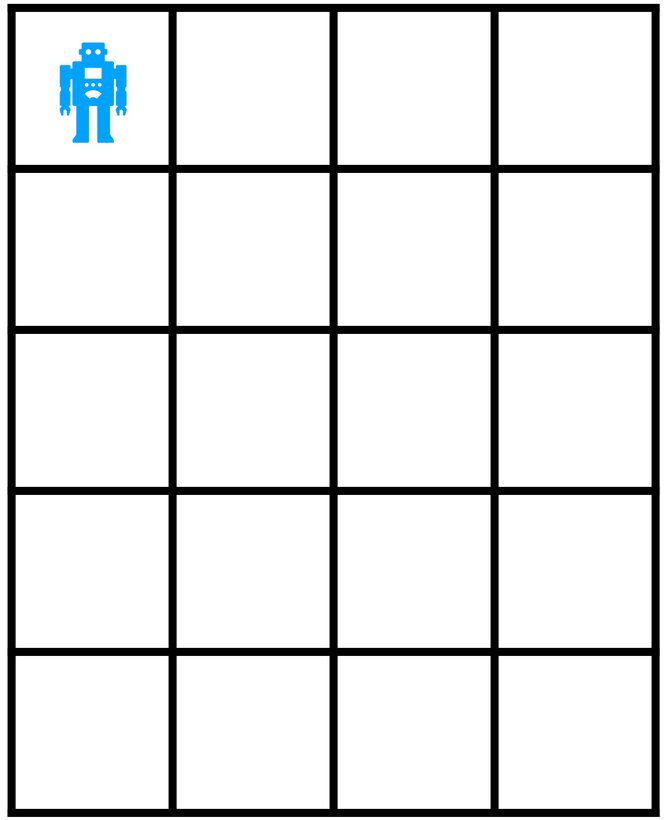
Limitations of two-stage approach

- **Supervised learning doesn't know about the decision-making**
  - Its objective is, for example, to maximize accuracy
- **For decision making, different errors have different costs**
  - E.g., missing the cliff could have dire consequences. missing sky less so.
  - Incorporating these costs into the learning objective is tough
- **Several other limitations:**
  - need labeled data
  - improvements in SL do not necessarily lead to improvements in decision making
  - …

Alternative: Reinforcement learning (RL)

- **Incorporates both stages in a single framework**
- Incorporates the ideas of:
  - **state (observation)**
  - **action**
  - **reward**

Initial example with grid world



- Each cell is a state (S)
- Actions indicate which movements are possible:

$$A := \{L, R, U, D\}$$

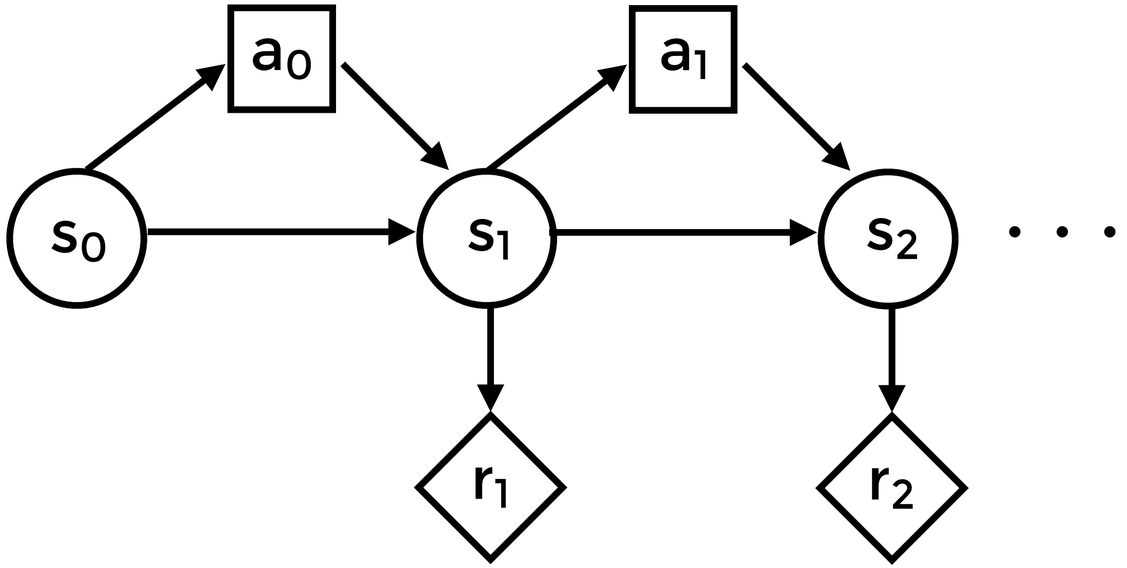
- Rewards encode the task: $R(s)$
- Transition probabilities encode the outcome of an action: $P(s' \mid s, a)$

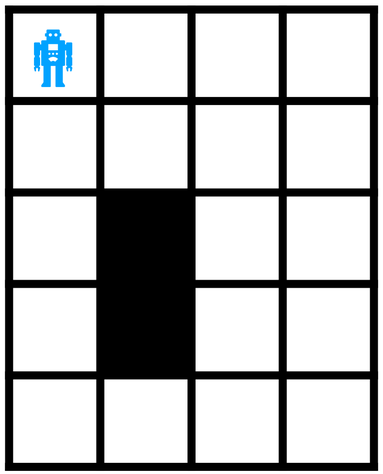
This week we discuss a version of RL where these are observed

Markov Decision Process (MDP)

- **Provide a framework for decision-making under uncertainty**
  - **Markov process with decisions and utilities**
  - **Assumes stationarity (i.e., transitions are fixed across time)**

- Square nodes: decisions
- Circle nodes: States
- Diamond nodes: utility





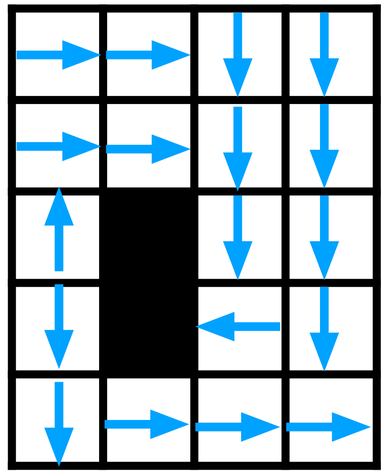
Markov Decision Process (MDP)

$\langle A, S, P, R, \gamma \rangle$

- **A: set of actions**
- **P(S' I S,A): transition probabilities**
- **R(S): reward function**
- γ **: discount factor** $\in [0, 1]$

- **A policy:** $\pi : S \rightarrow A$
- **Goal: find the optimal policy**

π



Optimal policy?

- Agent is trying to maximize its rewards (utility)
  - Utility simply assigns a real value to a state
  - Typically combine rewards with an additive function

$$\sum_t R(s_t)$$

Discounting (γ)

- **The sum of rewards could be infinite/unbounded**

$$\lim_{T \to \infty} \sum_{t}^{T} R(s_t)$$

- **A typical solution is to use a discount factor** $0 \leq \gamma \leq 1$

$$\lim_{T \to \infty} \sum_{t}^{T} \gamma^t R(s_t)$$

  - **Geometric series. Bounded by:** $\frac{R_{max}}{1-\gamma}$
  - **Intuition: would rather have rewards sooner**

Solving an MDP

- **Find the optimal policy of an MDP**

$$\pi^*(s) \quad \forall s$$

- **Policies are evaluated using their expected utility:**

$$EU(\pi) = \sum_{t=0}^{\infty} \gamma^t \sum_{s_{t+1}} P(s_{t+1} \mid s_t, \pi(s_t)) R(s_{t+1})$$

- **The optimal policy is the one with highest expected utility:**

$$EU(\pi^*) \geq EU(\pi) \quad \forall \pi$$

Solving an MDP

1. Value iteration
2. Policy Iteration

Value Function

- $V(s_t)$: **The value of being in state s at time t**

$$V(s_t) := \text{expected sum of rewards of being in s}$$

Finite horizon

- **Assume that the process has T steps**
- **The value at step T is** $V(s_T) = R(s_T)$
- **The value at step T-1 is**

$$V(s_{T-1}) = \max_{a_{T-1}} \left\{ R(s_{T-1}) + \gamma \sum_{s_T} P(s_T \mid s_{T-1}, a_{T-1}) R(s_T) \right\}$$

- **The value at step t is** $(0 \leq t \leq T)$

$$V(s_t) = \max_{a_t} \left\{ R(s_t) + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) V(s_{t+1}) \right\}$$

Bellman equation

- **Value of state s**

$$V(s_t) = \max_{a_t} \left\{ R(s_t) + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) V(s_{t+1}) \right\} \quad \forall s$$

  - **Recursive equations**
  - **The value of a state only depends on the state's reward and the neighbours' value**
  - **This is also known as a dynamic programming equation**

Value iteration (VI)

- Iteratively update V(s) for each state until convergence
- (Initialize V(s) for every state)
- For i=1,2,3,…
  - For s=1,…,S

$$V(s) = \max_a \left\{ R(s) + \gamma \sum_{s'} P(s' \mid s, a) V(s') \right\}$$

- The policy is implicit
  - Once converged: $\pi^*(s) = \arg\max_a \left\{ R(s) + \gamma \sum_{s'} P(s' \mid s, a) V^*(s') \right\} \ \forall s$

Policy Iteration (PI)

- **Improve policy explicitly.**

Start with any (e.g., random) policy π

Iterate until convergence:

1. **Given current policy get the value of each state**

Policy Evaluation

$$V^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) V^{\pi}(s') \qquad \forall s$$

2. **Update the current policy**

Policy Update

$$\pi'(s) = \arg\max_{a} \left\{ R(s) + \gamma \sum_{s'} P(s' \mid s, a) V^{\pi}(s') \right\} \quad \forall s$$

PI vs. VI

- **Value iteration is faster per iteration**
- **Policy iteration converges in fewer iterations**