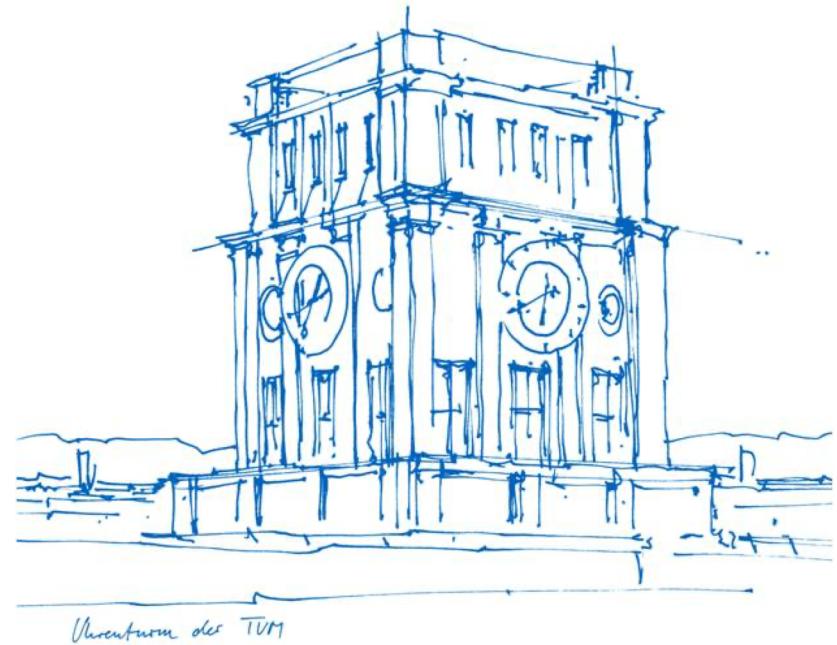


# Exercises for Social Gaming and Social Computing (IN2241 + IN0040) – Introduction to

## Exercise Sheet 3 Collaborative Filtering with Steam Games



# Exercise Content

---

## 1. Introduction to Python and Network Visualization

June 6-12

Python basics and first look at network analysis and visualization with the Simpsons and NetworkX

## 2. Social Network Analysis: Comparison of Centrality Measures

June 13-19

Deepen knowledge about centrality measures by comparing different measures using NetworkX and your own implementations.

## 3. Social Recommender Systems

June 20-26

Investigate variations of the famous Collaborative Filtering algorithm for recommending games using data from Steam.

## 4. Sentiment Analysis on DotA with Linear Regression

June 27-July 3

Analyze data from the game DotA in terms of sentiment.

## 5. Clustering: Identifying Player Types in WoW

July 4-10

Using data from WoW we aim at comparing different clustering paradigms (etc. GMMs or DBSCAN) for identifying player types

## 6. NLP on Social Media: Climate Change Activists vs. Deniers

July 11-17

With the help of modern NLP techniques and deep learning models, we analyze a Twitter dataset on Climate change

## Exercise Sheet 3: Collaborative Filtering with Steam Games

---

- **goal**: look at the games your friends **play the most** to figure out which ones you would love to play yourself
- build a **collaborative filtering** recommender system using the data we gather from your own **Steam profile**
- collaborative filtering is based on a **rating** to determine the similarity between users.
- **playtime** is usually the most authentic **metric** of enjoyment for gamers
  - therefore, we will use playtime as our rating system.

## The Data: Your friends' Steam library

---

- use **steam Web-API** to fetch the games your friends and *their* friends play.
- if you prefer not to use your own **Steam ID**, you can use the default ID we provide
  - we recommend to use your **own** ID though

## Tasks

---

### Task 3.1: Obtaining the data

1. Your first task is to **gather the data** needed to create the recommender system. **Create a data structure** that holds the needed information for each player and game. To do this, **open the URL** with the given `request()` function, **read** the json response and retrieve your games library and playtime. Then **save** the games into a dictionary with `key=name` and `values=playtime`. **Do not add** games with 0 playtime to this dictionary.

#### Notes:

- You have three different options to solve this exercise. You can either:
  - Use your own Steam profile (strongly recommended)
  - Use the provided default Steam account (in case you do not own a Steam profile)
  - Use the provided .json file (in case you do not have a Steam profile and the default Steam account becomes overcrowded)
- your choice will not affect your grade in any way
- You cannot obtain a list from your profile with the Steam API unless your profile is set to public.
- Upon executing the code below, you will notice that a lot of profiles " `couldn't decode` ". These are private or deleted profiles and it is totally fine to get this message.

# Tasks (cont.)

---

## Hints:

- In case you wish to use your own Steam profile, but are afraid to share your personal key and id, please be informed that you can delete them **after** solving the tasks and before submitting your solutions. The outputs will be saved in the Jupyter Notebook.
- To obtain the games a user owns, use this: `games = data['response'][‘games’]`. This returns a list of games, including the playtime (in minutes) which can be retrieved like this: `playtime = game[‘playtime_forever’]`, where game refers to an item from the list of games.

## Task 3.2: Association rule mining

The concept of association rule mining is rather simple: Looking at an itemset, one tries to find dependencies between items that could "belong together". A common example would be buying food at the store: If, for example, meat and salt are bought together often, but meat without salt not that often, it is assumed that there is a connection between those two. For games, if it was found that most of the users who own the demo version of a game also own the full version of that game, it would be a reasonable assumption that these users liked the demo and therefore bought the full version.

1. Your task here is to first convert the dictionary you created into a list of lists as this is the input required for the algorithm to work. Then, print out the most frequent items using the min\_support attribute. Finally, print out the association rules and play around with the threshold value to get a reasonable amount of rules.

2. Discuss your results and try to answer the following questions: What kind of recommendations can be made? What does a confidence of 1.0 mean and is it meaningful for recommending games? Can you spot a correlation between the games with the highest support and the rules with the highest confidence? How does this affect the lift?

**Hint:** Play around with the threshold values until you get a reasonable amount (4-30) rows as output.

# Association rules

---

- set of users and a set of games
- each user likes a subset of games :

User id	Game g <sub>1</sub>	Game g <sub>2</sub>	Game g <sub>3</sub>	Game g <sub>4</sub>	Game g <sub>5</sub>	
1	yes		yes			
2		yes	yes		yes	
3		yes		yes		
4		yes			yes	

- Association Rule: if set of games X is liked by the users then a set of games Y is also liked  
(Shopping basket analysis: users = shopping baskets; games = items)

# Association rules

- The most important metrics for rules are **support**, **confidence** and **lift**.
- **Support:** the proportion of users that like game-set

$$supp(X) = \frac{|\{u_i | X \subseteq G_i\}|}{|U|}$$

- **Confidence** of a rule  $X \rightarrow Y$  : proportion of users liking X that also like Y

$$conf(X \rightarrow Y) = \frac{supp(X \cup Y)}{supp(X)}$$

In terms of probabilities,  
this corresponds to  $P(Y|X)$

- **Lift** of a rule : mutual information of X and Y (how often do they appear together compared to how often that would be the case if they were independent .

$$lift(X \rightarrow Y) = \frac{supp(X \cup Y)}{supp(X) * supp(Y)}$$

In terms of probabilities of the events x: "X is liked" and y: "Y is liked" this  
would be  $\frac{P(x, y)}{P(x) * P(y)} = \frac{P(x \cap y)}{P(x) * P(y)}$  „pointwise mutual information“ (PMI))

# Association rules: example

Example database with 5 transactions and 5 items

transaction ID	milk	bread	butter	beer	diapers	eggs	fruit
1	1	1	0	0	0	0	1
2	0	0	1	0	0	1	1
3	0	0	0	1	1	0	0
4	1	1	1	0	0	1	1
5	0	1	0	0	0	0	0

Example of Support, and support X confidence

if Antecedent then Consequent ↴	support ↴	support X confidence ↴
if buy milk, then buy bread	2/5= 0.4	0.4X1.0= 0.4
if buy milk, then buy eggs	1/5= 0.2	0.2X0.5= 0.1
if buy bread, then buy fruit	2/5= 0.4	0.4X0.66= 0.264
if buy fruit, then buy eggs	2/5= 0.4	0.4X0.66= 0.264
if buy milk and bread, then buy fruit	2/5= 0.4	0.4X1.0= 0.4

$$\{\text{milk, bread}\} \Rightarrow \{\text{butter}\} \text{ has a lift of } \frac{0.2}{0.4 \times 0.4} = 1.25.$$

lift = {

- =1 → indep. → no useful rule
- >1 → useful rule
- <1 → items are substitutes → no useful rule

# Tasks (cont.)

## Task 3.3: The Recommender System: Similarity Score

Finally, it is time to build the recommender system.

1. The first thing to do is to **implement a similarity score** that will be used to predict a user's playtime of an unowned game. We implement a similarity score between two users by taking the relative distance between two players. We use the following formula:

$$d(u, v) = \sum_{i \in \text{common games}} \frac{|r_{u,i} - r_{v,i}|}{r_{v,i}}$$

Where  $u$  and  $v$  are users and  $r_{u,i}$  is the playtime of user  $u$  for game  $i$ .

You can then return the similarity with

$$w_{u,v} = \frac{1}{1 + d(u, v)}$$

### Notes:

- If no common games exist return 0.

a) **Implement similarity scores:** Besides the given similarity score, we want to explore how other measurements behave. Hence, we will implement the euclidean distance and cosine similarity. The scores can be selected by setting the respective variable on `True`.

# Tasks (cont.)

## Task 3.4: Recommender System: Predict ratings

With the similarity score calculated, we can now predict a user's playtime for games they don't own.

1. First, we **create a set of all games**, but we **delete** all games that are owned by less than 3 players. The reason is simple: If only 1 or 2 players own a game, it is impossible to derive a meaningful prediction since there is not enough data.

The predicted playtime for a game works analogous to the predicted rating of a movie/item in a conventional collaborative filtering recommender system:

$$r_{u,i} = \frac{\sum_{v \in N_i(u)} w_{u,v} r_{v,i}}{\sum_{v \in N_i(u)} w_{u,v}}$$

where

- $r_{u,i}$  is the estimated recommendation of item  $i$  for target user  $u$ .
- $N_i(u)$  is the set of similar users to target user  $u$  for the designated item  $i$ .
- $w_{u,v}$  is the similarity score between users  $u$  and  $v$  (used as a weighting factor).

### Notes:

- In our case, we use playtime as a recommendation measure and the set  $N_i(u)$  consists of user  $u$  friends list and friends of friends list. In our scenario, we do not need the index  $i$  as our friends list does not change between games.
- Keep in mind that we have already taken out the games with a playtime of 0. In this case, they are considered "unowned" and not taken into account in this exercise.

# Collaborative Filtering

## Collaborative Filtering (cont.)

				items				
				—	—	1	2	—
				—	—	1	—	—
				?	—	1	—	5
				9	—	—	—	—
				—	3	3	—	—
				6	—	—	—	5
				—	3	—	2	—
				—	—	0	—	—
				6	—	—	—	6
				—	—	9	4	—
				—	4	—	—	6

## Collaborative Filtering (cont.)

				items				
				—	—	1	2	—
				—	—	1	—	—
				?	—	1	—	5
				—	—	—	—	—
				—	3	3	—	—
				—	—	—	5	
				—	3	—	2	—
				—	—	0	—	—
				—	—	—	6	
				—	—	9	4	—
				—	4	—	—	6

## Collaborative Filtering (cont.)

items

		users				items				
		user u		item i						
r =	4 - - -		- - 1 2 -		- 1 - 5					
	- - -	5	-	-	1	-	-	-	-	-
	-	5 5 -	?	-	1	-	5	-	-	-
	-	9 - -	9	-	-	-	-	-	-	-
	-	- - -	-	3	3	-	-	-	-	-
	-	8 5 -	6	-	-	-	5	-	-	-
	1	- - 1	-	3	-	2	-	-	-	-
	-	- - -	-	-	0	-	-	-	-	-
	-	7 6 1	6	-	-	-	6	-	-	-
	-	- - 1	-	-	9	4	-	-	-	-
	-	- - -	-	4	-	-	6	-	-	-

user u  
 item i

$\mathcal{N}_i(u)$   
 $\mathcal{N}_i(u)$ : users that rated item  $i$  and that are similar to user  $u$ , e.g.:  

$$\mathcal{N}_i(u) = \{v^{(i)} | sim(u, v^{(i)}) > \alpha\}$$

where e.g.

$sim(u, v^{(i)}) = w_{uv} =$   
 $\cos(\mathbf{u}, \mathbf{v}^{(i)}) \sim \mathbf{u} * \mathbf{v}^{(i)}$

or:

$sim(u, v^{(i)}) = w_{uv} =$   

$$\frac{\mathbf{1}}{1 + |\mathbf{u} - \mathbf{v}^{(i)}|^2}$$

## Collaborative Filtering (cont.)

				items					
				—	—	1	2	—	
4		—	—	—	—	—	1	2	—
—		—	—	5	—	—	1	—	—
—	5	5	—	?	—	1	—	5	user u
—	9	—	—	9	—	—	—	—	

now: predicted rating for item i of user u

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in \mathcal{N}_i(u)} w_{uv} (r_{vi} - \bar{r}_v)}{\sum_{v \in \mathcal{N}_i(u)} w_{uv}}$$

or (more simple):

$$\hat{r}_{ui} = \frac{\sum_{v \in \mathcal{N}_i(u)} w_{uv} r_{vi}}{\sum_{v \in \mathcal{N}_i(u)} w_{uv}}$$

$\mathcal{N}_i(u)$ : users that rated item i and that are similar to user u, e.g.:

$$\mathcal{N}_i(u) = \{v^{(i)} | sim(u, v^{(i)}) > \alpha\}$$

where e.g.

$$sim(u, v^{(i)}) = w_{uv} = \cos(\mathbf{u}, \mathbf{v}^{(i)}) \sim \mathbf{u} * \mathbf{v}^{(i)}$$

or:

$$sim(u, v^{(i)}) = w_{uv} = \frac{1}{1 + |\mathbf{u} - \mathbf{v}^{(i)}|^2}$$

# Tasks (cont.)

---

## Task 3.5: Recommender System: Discussion

1. Sort the predicted ratings by estimated playtime (highest first) and print out the top 8 predictions for you (or the default user if you are using the default ID).
2. Discuss the difference in recommendations between the collaborative filtering approach and the association rule approach. Would you consider one more accurate than the other? Why/why not?
3. Discuss the differences in the similarity scores.

## Submitting your solution

---

- work by **expanding** the .ipynb iPython notebook for the exercise that you **downloaded** from Moodle
- **save** your expanded .ipynb iPython notebook in **your working directory**
- **submit** your .ipynb iPython notebook **via Moodle** (nothing else)
- remember: working in groups is not permitted. Each student must submit **their own** .ipynb notebook!
- we check for **plagiarism**. Each detected case will be graded with 5.0 for the whole exercise
- **deadline**: check Moodle



# Literature

---

- Desrosiers, C., & Karypis, G.: A Comprehensive Survey of Neighborhood-based Recommendation Methods, 2011 in: Ricci et al (eds.) "Recommender Systems Handbook", Springer 2011