

# Computational Communication Science 2

## Week 6 - Lecture

### »Recommender systems«

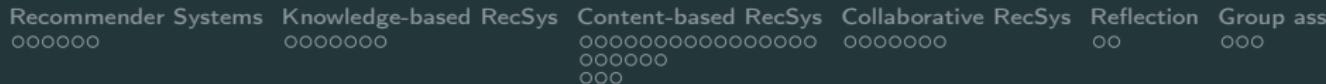
---

Anne Kroon

a.c.kroon@uva.nl, @annekroon

May, 2022

Digital Society Minor, University of Amsterdam



# Today

Recommender Systems

Knowledge-based Recommender Systems

Content-based Recommender Systems

Collaborative Recommender Systems

The role of recommender systems in society

Group Assignment

Recommender Systems

oooooo

Knowledge-based RecSys

oooooooo

Content-based RecSys

oooooooooooooooooooo

Collaborative RecSys

ooooooo

Reflection

oo

Group ass

ooo

**Updated exam planning; The take-home  
exam will be published on 30 May at 9 am.  
The deadline for submitting your answers is  
on 2 June at 9 am.**



*Everything clear from last  
weeks?*

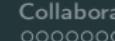
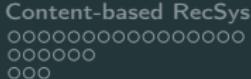
## Main points from last week

I assume that by now, everybody knows:

- how to clean and preprocess textual data;
- how to vectorize the data;
- how to calculate cosine similarity using different techniques;

# Recommender Systems

---





Recommender Systems in Communication Science

## New research questions

1. Political communication and journalism. E.g., to craft personalized news diets. This may have, however, consequences for the diversity of news diets and for democracy (Locherbach & Trilling, 2018; Möller et al., 2018)
  2. Organizational and corporate communication. E.g., applications in the field of hiring and recruitment.
  3. Persuasive communication. E.g., in the health domain—recommendation algorithms for tailored health interventions (Kim et al., 2019)
  4. Entertainment communication. E.g., movie recommenders

# Recommender Systems

## Two central problems within Recommender Systems

1. **Predicting problem.** Typical problem involves a lot of missing data (e.g., user only rated a small subset of movies/news articles/ etc.) How can we deal with missing data?
2. **Ranking problem.** Given  $n$  items, can we identify the top  $k$  items to recommend?

These problems are not isolated; rather, they are connected.

# Recommender Systems

## How do recommender systems learn?

1. **Explicit user preferences.** Ratings or responses
2. **Implicit user preferences.** E.g., clicks or viewing time
3. **Content.** E.g., based on text similarity techniques

# Recommender Systems

Types of recommender systems (Locherbach & Trilling, 2018;  
Möller et al., 2018; Wieland et al., 2021)

1. 'Basic' knowledge-base recommender systems
2. Content-based recommender systems
3. Collaborative recommenders

# Knowledge-based RecSys

---

## Knowledge-based recommender system

### When to use?

- To overcome the **cold start problem**; when we do not have ratings of individual users.
- Simple model. It does not rely on user's explicit or implicit ratings, but on specific queries.
- Typical use case: Real-estate. Buying a house is, for most families, a rare/ single event.

**funda**

Het geluk van een sneeuwpop  
in je eigen tuin

Koop   Huur   Nieuwbouw   Recreatie   Europa   Bedrijfsruimte

Plaats, buurt, adres, etc. + 0 km Van € 0 Tot Geen maximum Zoek

## Use case: IMDb database

	genres	title	tagline	release_date	vote_average	vote_count
0	[action, adventure, fantasy, science fiction]	Avatar	Enter the World of Pandora.	2009-12-10	7.2	11800
1	[adventure, fantasy, action]	Pirates of the Caribbean: At World's End	At the end of the world, the adventure begins.	2007-05-19	6.9	4500
2	[action, adventure, crime]	Spectre	A Plan No One Escapes	2015-10-26	6.3	4466
3	[action, crime, drama, thriller]	The Dark Knight Rises	The Legend Ends	2012-07-16	7.6	9106
4	[action, adventure, science fiction]	John Carter	Lost in our world, found in another.	2012-03-07	6.1	2124



*What are relevant variables to  
use in a knowledge-based  
recommender system?*

## Knowledge-based recommender system

How can we work with user input without a front-end (such as the website of funda)? → enter python's native `input()` function.

```
print("What is your favorite movie genre?")
genre = input()
```

```
1 what is your favorite movie genre?
2 [...]
```

## Improving knowledge-based recommender system

### When to use?

- It is important to think about ways to make the recommendation relevant for individuals
- Do you have more information in your db that make your top-listed recommendations as relevant as possible?

```
recommend_movies = movies.sort_values('vote_average', ascending=False)
```

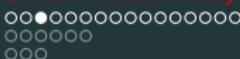
# **Content-based RecSys**

---



## Content-based systems

- Recommends items based on user's profiles.
- Profiles are based on e.g., ratings, and represents user's tastes/preferences.
  - For example, how often a user has clicked on, or liked, a movie.
- Recommendation is based on **similarity** between items in the content.
  - Content is here: e.g., genre, tags, plot, authors, directors, location, etc.



# Example of a content-based recsys

imdb.com/title/tt0241527/

Cast & crew · User reviews · Trivia · IMDbPro · All topics

Play trailer 0:32

13 VIDEOS

99+ PHOTOS

Adventure · Family · Fantasy

An orphaned boy enrolls in a school of wizardry, where he learns the truth about himself, his family and the terrible evil that haunts the magical world.

Director Chris Columbus

Writers J.K. Rowling (novel) · Steve Kloves (screenplay)

Stars Daniel Radcliffe · Rupert Grint · Richard Harris

Watch on Prime Video · Rent/buy from EUR 2.99

+ Add to Watchlist

1.8K User reviews · 274 Critic reviews · 65 Metascore

IMDbPro See production, box office & company info

# Example of a content-based recsys

## More like this



★ 7.5



Harry Potter and the  
Chamber of Secrets

[Watch options](#)

★ 7.9



Harry Potter and the  
Prisoner of Azkaban

[Watch options](#)

★ 7.7



Harry Potter and the  
Goblet of Fire

[Watch options](#)

★ 7.5



Harry Potter and the  
Order of the Phoenix

[Watch options](#)



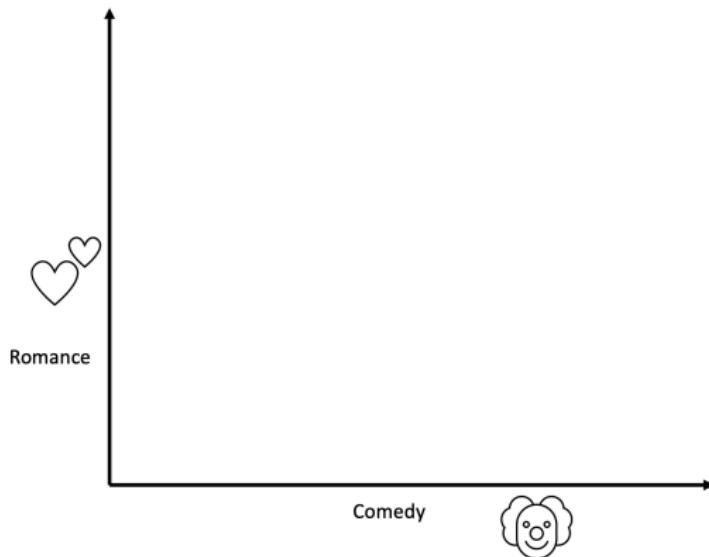
## Use case: IMDb database

Let's have a look at our use case again.

Are there attributes that you could use to estimate similarity in movies?

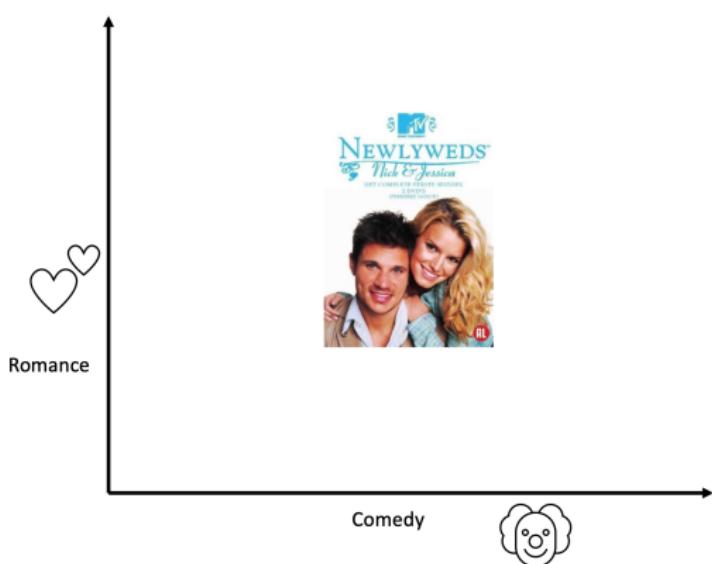
	genres	title	tagline	release_date	vote_average	vote_count
0	[action, adventure, fantasy, science fiction]	Avatar	Enter the World of Pandora.	2009-12-10	7.2	11800
1	[adventure, fantasy, action]	Pirates of the Caribbean: At World's End	At the end of the world, the adventure begins.	2007-05-19	6.9	4500
2	[action, adventure, crime]	Spectre	A Plan No One Escapes	2015-10-26	6.3	4466
3	[action, crime, drama, thriller]	The Dark Knight Rises	The Legend Ends	2012-07-16	7.6	9106
4	[action, adventure, science fiction]	John Carter	Lost in our world, found in another.	2012-03-07	6.1	2124

## Similarity between movies





## Similarity between movies

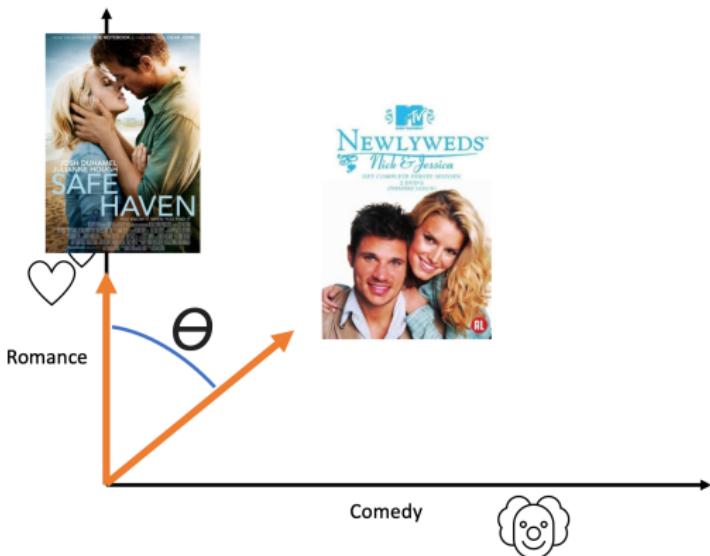




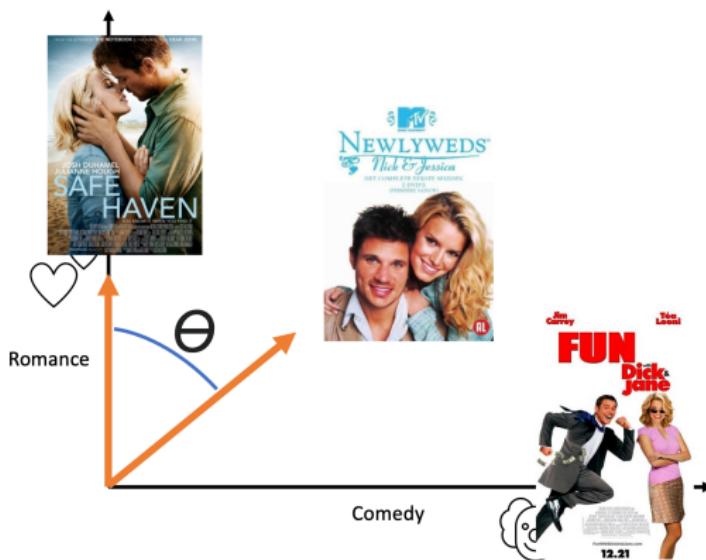
## Similarity between movies



## Similarity between movies



## Similarity between movies



## Let's put this in code!

```
data = data.sample(6)  
data[['title', 'genres']]
```

Out[239]:

		title	genres
0		Avatar	action adventure fantasy science fiction
1	Pirates of the Caribbean: At World's End		adventure fantasy action
2		Spectre	action adventure crime
3		The Dark Knight Rises	action crime drama thriller
4		John Carter	action adventure science fiction

## Feature selection and vectorization

Let's assume we want to calculate similarity based on genres.  
Therefore, we need to vectorize this column.

```
tfidf = TfidfVectorizer(stop_words='english')  
tfidf_matrix = tfidf.fit_transform(data['genres'])
```

Note that we use **tfidf vectorizer** here, but you might opt for a different one.



## Calculate similarity

Now, let's calculate cosine similarity scores between the genre attributes of the movies

```
from sklearn.metrics.pairwise import cosine_similarity  
sim = cosine_similarity(tfidf_matrix)
```

This returns an array of the similarity scores between each movie and all other movies.

## Cosine Similairty

Let's inspect the output...

```
print(sim)
[[1.          0.34503493 0.          0.          0.40824829 0.          ]
 [0.34503493 1.          0.16581288 0.          0.28171984 0.29130219]
 [0.          0.16581288 1.          0.25964992 0.          0.56921261]
 [0.          0.          0.25964992 1.          0.44115109 0.4561563 ]
 [0.40824829 0.28171984 0.          0.44115109 1.          0.          ]
 [0.          0.29130219 0.56921261 0.4561563  0.          1.          ]]
```

		title	Avatar	Pirates of the Caribbean: At World's End	Spectre	The Dark Knight Rises	John Carter
	genre	action adventure fantasy science fiction	adventure fantasy action		action adventure crime	action crime drama thriller	action adventure science fiction
	title	genres					
Avatar	action adventure fantasy science fiction	1.000000	0.691870	0.315126	0.084696	0.859850	
Pirates of the Caribbean: At World's End	adventure fantasy action	0.691870	1.000000	0.455470	0.122417	0.366490	
Spectre	action adventure crime	0.315126	0.455470	1.000000	0.473354	0.366490	
The Dark Knight Rises	action crime drama thriller	0.084696	0.122417	0.473354	1.000000	0.098501	
John Carter	action adventure science fiction	0.859850	0.366490	0.366490	0.098501	1.000000	

Based on this overview, you may assume that users that like *Avatar*, may be interested in *John Carter*. If you want to convert output of `cosine_similarities` to a `df` type of object, see here

[https://github.com/annekroon/CCS-2/blob/main/week06/cosine\\_to\\_df.md](https://github.com/annekroon/CCS-2/blob/main/week06/cosine_to_df.md).



*Can we automate this process?  
Can we automatically find the  
entry with the most similar  
movie?*

# Example of a content-based recsys

What are interesting features that you can use to base your recommendation on?

## Get the most similar movies

First, create an array of cosine scores

```
1 data['combined_features'] = data[['original_title', 'genres',
2     'overview', 'tagline']].apply(lambda x:
2     ','.join(x.dropna().astype(str)),axis=1)
3 tfidf = TfidfVectorizer(stop_words='english')
4 tfidf_matrix = tfidf.fit_transform(data['combined_features'])
5 cosine_sim = cosine_similarity(tfidf_matrix)
```

Let's say we want to find the similarity scores associated with the movie The Dark Night Rises.

## Example of a content-based recsys

Let's say we want to find the similarity scores associated with the movie The Dark Night Rises. Just by looking at the dataframe, we know it is at index 3.

```
1 cosine_sim[3]
```

```
array([0.02331349, 0.00399197, 0.00780174, ..., 0.02860083, 0.03756737, 0.018984 ])
```

A more systematic way to get the index value, is to simple look it up:

```
1 indices = pd.Series(data.index, index = data['original_title'])
2 index = indices['the dark knight rises']
3 print(index)
```

3

## Get the most similar movies

- Next, we need to sort the associated vector of cosine similarity scores for this movie to get the most similar movies.
- Before sorting the cosine scores, we map the movie-index to the cosine value. We can do so by simple enumerating the cosine scores:

```
1 sim_scores = list(enumerate(cosine_sim[index]))
```

```
[(0, 0.023313489832942038),  
(1, 0.003991972883233364),  
(2, 0.007801739082093903), ...]
```

Next, we can simply sort the cosine scores:

```
1 sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)  
2 sim_scores = sim_scores[1:11]
```

```
[(3, 0.9999999999999998),  
(65, 0.3484608502957436), ...]
```

## Get the most similar movies

And finally look up the associated movies..

```
1 movie_indices = [i[0] for i in sim_scores]  
2 movie_indices
```

```
[299, 1359, 3854, 428, 119, 210, 9, 2507, 979]
```

Map the index values to the dataframe:

```
1 data.iloc[movie_indices]['title']
```

299	Batman Forever
1359	Batman
3854	Batman: The Dark Knight Returns, Part 2
428	Batman Returns
119	Batman Begins
210	Batman & Robin
9	Batman v Superman: Dawn of Justice
2507	Slow Burn



## Feature selection and engineering

- Feature engineering is very important here. What qualities or attributes do we want to incorporate?
- Think about this critically. Instead of using genres or authors, you could think about crafting features yourself, such as topics.

how can we identify similar items?

1. Cosine similarity
2. Soft cosine similarity
3. LDA: topic scores

## Benefits

- Content-based recommender systems can be very efficient...
- They are often part of more complex recommender systems that leverage (deep) supervised learning

## Drawbacks

- Features that are not part of the user profile will be neglected; e.g., if the user does not like Super Hero movies, the recommender system will never recommend this.
- does not use the power community data. Recommendations may be obvious (e.g., *Harry Potter* recommendation when you like *Lord of the Rings*)

# Collaborative RecSys

---

# Collaborative filtering

## What are collaborative systems?

- Tries to overcome some of the limitations of content-based systems
- Leverages the power of the community, tries to give relevant, but also surprising recommendations.
- Very successful models in industry settings

## Types of collaborative systems

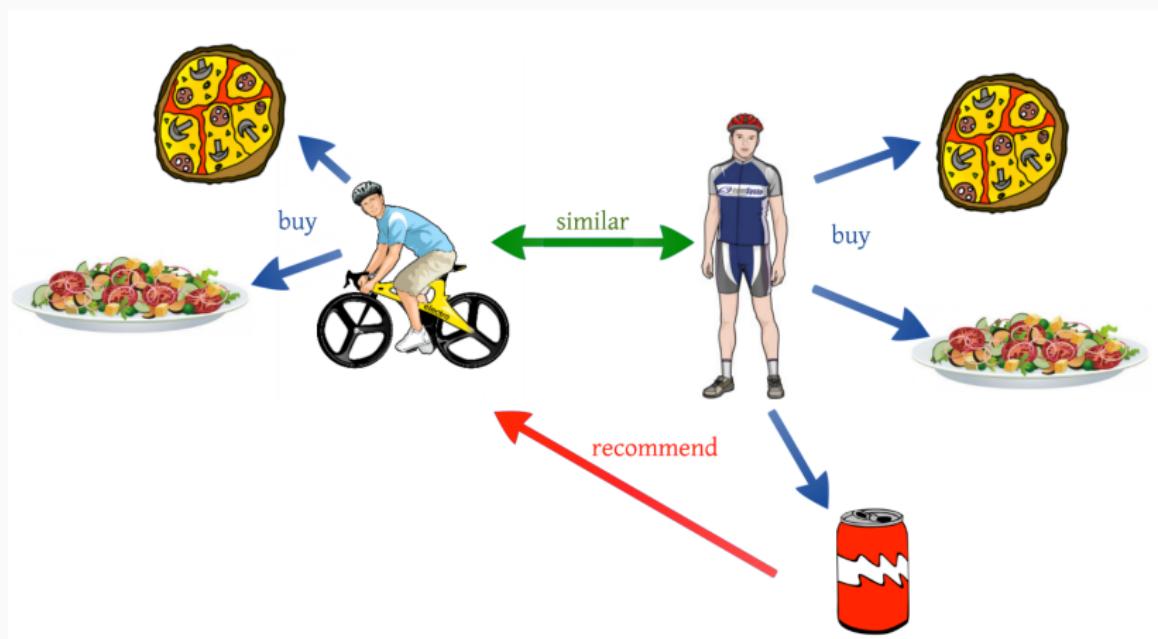
1. User-based collaborative filtering
2. Item-based collaborative filtering

## User-based filtering

### Similar users...

- If we find users that are similar in terms of the things they liked in the past...
- ...we can use this information to predict what they like in the future

## User-based filtering



1

<sup>1</sup>Source:

<https://medium.com/@akhilesh3091999/recommender-system-bb032b16ab67>



## Item-based filtering

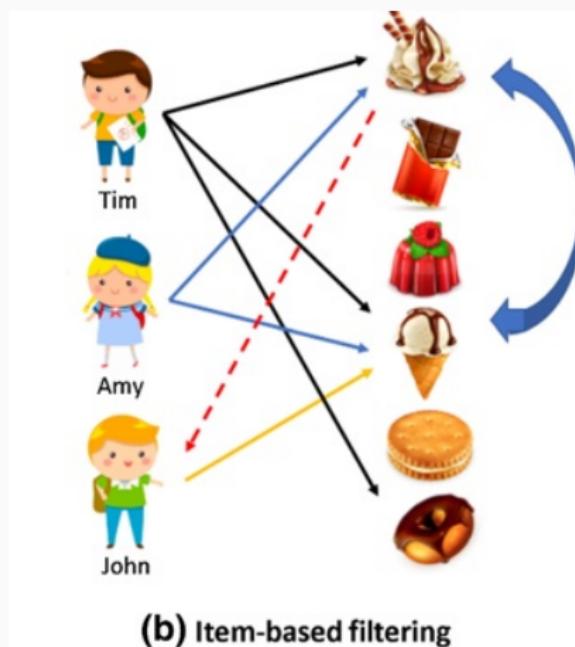
### Similar items...

- If two items are rated similarly by a group of people
- ...these products might be similar

### Example...

- If Alex, Sanne and Marthe like skincare product  $X$  and  $Y$ , and Denise buys skincare product  $Y$ ,  $X$  will be recommended.

## User-based filtering



2

<sup>2</sup>Source:<https://predictivehacks.com/item-based-collaborative-filtering-in-python/>



# Example: Amazon

**Recently Bought Together**

Price for all three: **\$74.26**

Add all three to Cart | Add all three to Wish List | Show availability and shipping details

This Item: Beginning Ruby: From Novice to Professional [Expert's Voice in Open Source] by Peter Cooper Paperback \$27.79

Learn to Program, Second Edition (The Facets of Ruby Series) by Chris Pine Paperback \$16.94

Ruby on Rails Tutorial: Learn Web Development with Rails (2nd Edition) (Addison-Wesley Professional Ruby...) by Michael Hartl Paperback \$29.48

**Customers Who Bought This Item Also Bought**

Learn to Program, Second Edition (The Facets of Ruby) by Chris Pine Paperback \$16.94 ✓Prime	The Well-Grounded Rubyist by David A. Black Paperback \$32.49 ✓Prime	Ruby on Rails Tutorial: Learn Web Development... by Michael Hartl Paperback \$29.48 ✓Prime	The Ruby Programming Language by David Flanagan Paperback \$29.39 ✓Prime	The Well-Grounded Rubyist by David A. Black Paperback \$29.67 ✓Prime
★★★★★ 42	★★★★★ 39	★★★★★ 70	★★★★★ 74	★★★★★ 19 <b>#1 Best Seller</b> in Ruby Programming Computer

## Reflection

---

## Reflection on the role of recommender systems in society

### Food for thought

- Consequences of recommender systems for democratic values in society

## Group ass

---



## Build your own recommender system

Part of the assignment: Build a recommender (20% of final grade)

- Build a recommender system, based on the insights from this week (week 6). It's up to you to decide whether you build a knowledge-based or content-based recommender system.
- Think about relevant features that you want to use in your algorithm design. Based on which features do you want to recommend content?



## Build your own recommender system

### Practice with the materials!

- To be able to do this correctly, it is essential that you understand the code of this week's lab session.
- Carefully walk through this week's assignment, and to whether questions arise.
- It's up to you to decide whether you want to build a simple knowledge-based or content-based recommender system. Base your selection on the available data columns.

## References

# References

---

-  Kim, H. S., Yang, S., Kim, M., Hemenway, B., Ungar, L., & Cappella, J. N. (2019). An experimental study of recommendation algorithms for tailored health communication. *Computational Communication Research*, 1(1), 103–129. <https://doi.org/10.5117/ccr2019.1.005.sukk>
-  Locherbach, F., & Trilling, D. (2018). 3bij3: A framework for testing effects of recommender systems on news exposure. *Proceedings - IEEE 14th International Conference on eScience, e-Science 2018*, 350–351. <https://doi.org/10.1109/eScience.2018.00093>
-  Möller, J., Trilling, D., Helberger, N., & van Es, B. (2018). Do not blame it on the algorithm: an empirical assessment of multiple recommender systems and their impact on content diversity. *Information Communication and Society*, 21(7), 959–977. <https://doi.org/10.1080/1369118X.2018.1444076>
-  Wieland, M., Von Nordheim, G., & Kleinen-Von Königslöw, K. (2021). One recommender fits all? An exploration of user satisfaction with text-based news recommender systems. *Media and Communication*, 9(4), 208–221. <https://doi.org/10.17645/mac.v9i4.4241>