Computational Communication Science 2
Week 3 - Lecture
»LDA and Recommender systems«

Anne Kroon

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

April 15, 2024

Digital Society Minor, University of Amsterdam

0

Today

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;

Topic modelling

Let's assume you want to know a bit more about the *content* you are investigating

Cosine and soft cosine do *not* inform us about substantive issues present in text.

1. Which topics can we extract from the corpus?
2. How present is each of these topics in each text in the corpus?

Recap: Document-term matrix

Document-term matrix

```
1       w1,w2,w3,w4,w5,w6 ...
2   text1, 2, 0, 0, 1, 2, 3 ...
3   text2, 0, 0, 1, 2, 3, 4 ...
4   text3, 9, 0, 1, 1, 0, 0 ...
5   ...
```

These can be simple counts, but also more advanced metrics, like tf-idf scores (where you weigh the frequency by the number of documents in which it occurs), cosine distances, etc.

- given a term-document matrix, easy to do with any tool
- probably extremely skewed distributions

Recap: clustering

- given a term-document matrix, we can easily find clusters of documents that resemble each other

We need other models to

1. model *simultaneously* (a) which topics we find in the whole corpus, and (b) which of these topics are present in which document; while at the same time
2. allowing (a) words to be part of multiple topics, and (b) multiple topics to be present in one document; and
3. being able to make connections between words "even if they never actually occured in a document together" (Maier et al., 2018)[p. 96]

Maier, D., Waldherr, A., Miltner, P., Wiedemann, G., Niekler, A., Keinert, A., ... Adam, S. (2018). Applying LDA Topic Modeling in Communication Research: Toward a Valid and Reliable Methodology. *Communication Methods and Measures, 12*(2–3), 93–118. doi:10.1080/19312458.2018.1430754

Topic modelling

An introduction to LDA

Enter **topic modeling with Latent Dirichlet Allocation (LDA)**

LDA, what's that?

No mathematical details here, but the general idea

- There are k topics, $T_1 \ldots T_k$
- Each document D_i consists of a mixture of these topics, e.g. $80\% T_1, 15\% T_2, 0\% T_3, \ldots 5\% T_k$
- On the next level, each topic consists of a specific probability distribution of words
- Thus, based on the frequencies of words in D_i, one can infer its distribution of topics
- Note that LDA is a Bag-of-Words (BOW) approach

Doing a LDA in Python

We will use gensim (Rehurek & Sojka, 2010) for this (make sure you have version >4.0)

Let us assume you have a list of lists of documents called `texts`:

```
1 print(texts[0][:115])
```

which looks something like:

```
1 'Stop the presses: CNN covered some actual news yesterday when it reported on the story of
↪  medical kidnapping victim Alyssa Gilderhus at the Mayo Clinic. But was it actually
↪  InfoWars and FreeMartyG which publicly shamed CNN into doing this real journalism? Cue the
↪  Mission Impossible theme music for this one...\n\nThis mission, as we accepted it, began
↪  more than a year ago during the baby Charlie Gard medical kidnapping scandal in the UK and
↪  we thought that it had ended with an apparently unsuccessf'
```

Řehůřek, R., & Sojka, P. (2010). Software framework for topic modelling with large corpora.

Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, pp. 45–50. Valletta,

Preprocessing

Your preprocessing steps and feature engineering decisions *largely* affect your topics

1. You can apply 'manual' preprocessing steps . . .
2. . . . In isolation or combination with for example `tfidf` transformations

```
1  texts_clean = [text.lower() for text in texts]
2  texts_clean=[" ".join(text.split()) for text in texts_clean]  #remove dubble spaces
3  texts_clean = ["".join([l for l in text if l not in punctuation]) for text in texts_clean]
   ↪  #remove punctuaction
4  texts_clean[0][:500]
```

which looks something like:

```
1  'stop the presses cnn covered some actual news yesterday when it reported on the story of
   ↪  medical kidnapping victim alyssa gilderhus at the mayo clinic but was it actually infowars
   ↪  and freemartyg which publicly shamed cnn into doing this real journalism cue the mission
   ↪  impossible theme music for this one this mission as we accepted it began more than a year
   ↪  ago during the baby charlie gard medical kidnapping scandal in the uk and we thought that
```

Preprocessing

Without *stopword removal*, *tfidf* transformation and/or *pruning*, you topics will not be very informative.

```
1 mystopwords = set(stopwords.words('english')) # use default NLTK stopword list;
  ↪ alternatively:
2 # mystopwords = set(open('mystopwordfile.txt').readlines())  #read stopword list from a
  ↪ textfile with one stopword per line
3 texts_clean = [" ".join(word for word in text.split() if word not in mystopwords) for text in
  ↪ texts_clean]
4 texts_clean[0][:500]
```

which looks something like:

```
1 'stop presses cnn covered actual news yesterday reported story medical kidnapping victim
  ↪ alyssa gilderhus mayo clinic actually infowars freemartyg publicly shamed cnn real
  ↪ journalism cue mission impossible theme music one mission accepted began year ago baby
  ↪ charlie gard medical kidnapping scandal uk thought ended apparently unsuccessful april
  ↪ fools joke cnn sure many recall charlie gard infant rare form otherwise notsorare
  ↪ condition mitochondrial disease story went viral made international news '
```

Tokenization

gensim expects a list of words (hence: tokenize your corpus)

```
1  tokenized_texts_clean = [TreebankWordTokenizer().tokenize(text) for text in texts_clean ] #
   ↪  tokenize texts; convert all strings to a list of tokens
2  tokenized_texts_clean[0][:500]
```

which looks something like:

```
1  ['stop',
2  'presses',
3  'cnn',
4  'covered',
5  'actual',
6  'news',
7  'yesterday',
8  'reported',
9  'story',
10 ..
```

LDA implementation

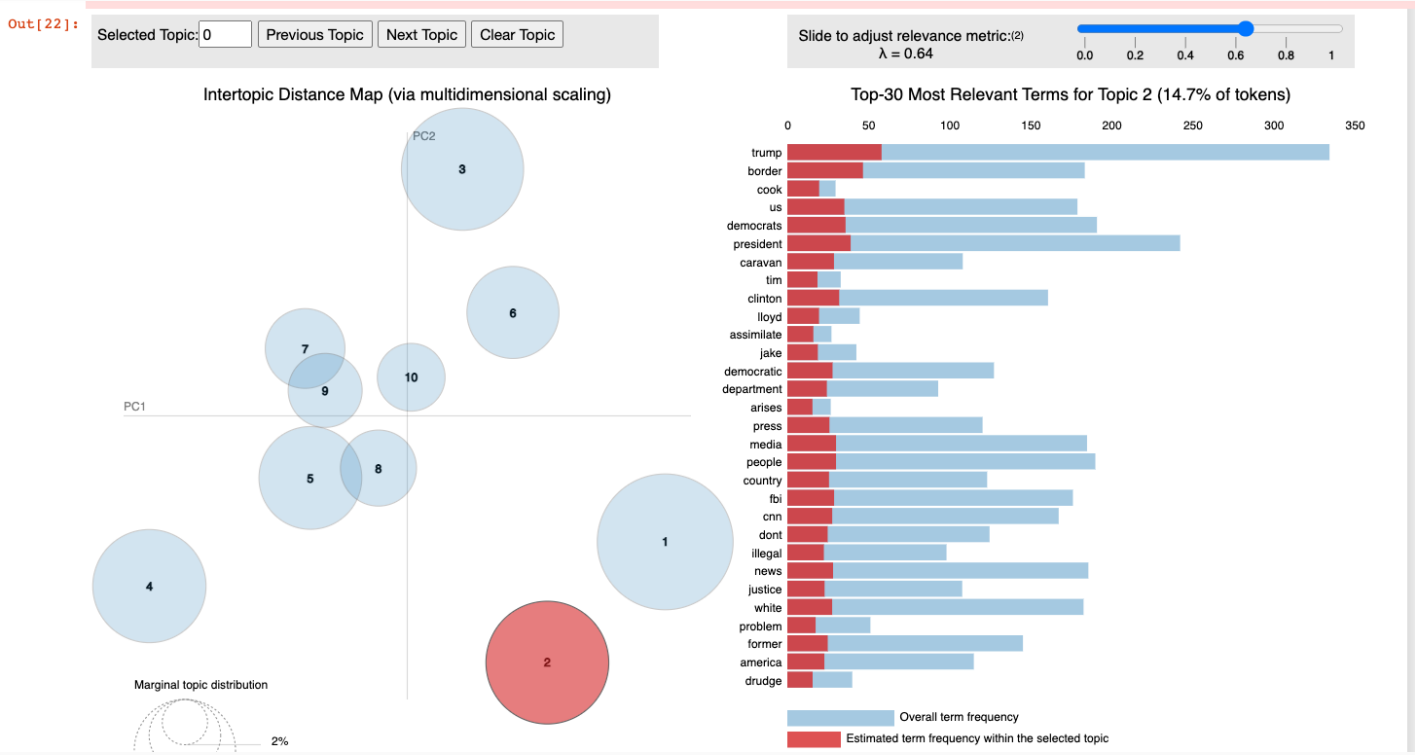
LDA implementation

```
raw_m1 = tokenized_texts_clean

# assign a token_id to each word
id2word_m1 = corpora.Dictionary(raw_m1)
# represent each text by (token_id, token_count) tuples
ldacorpus_m1 = [id2word_m1.doc2bow(text) for text in raw_m1]

#estimate the model
lda_m1 = models.LdaModel(ldacorpus_m1, id2word=id2word_m1, num_topics=10)
lda_m1.print_topics()
```

```
[(0,
'0.015*"trump" + 0.012*"said" + 0.006*"president" + 0.006*"people" + 0.004*"cnn" + 0.004*"us"
↪  + 0.004*"house" + 0.004*"news" + 0.003*"also" + 0.003*"twitter"'),
(1,
'0.010*"said" + 0.008*"trump" + 0.004*"one" + 0.004*"people" + 0.004*"us" + 0.004*"president"
↪  + 0.004*"would" + 0.003*"media" + 0.003*"also" + 0.003*"new"'),
(2,
'0.011*"trump" + 0.009*"said" + 0.007*"president" + 0.005*"would" + 0.004*"people" +
↪  0.004*"us" + 0.003*"also" + 0.003*"like" + 0.003*"news" + 0.003*"state"'),
(3,
'0.010*"trump" + 0.006*"president" + 0.005*"said" + 0.004*"would" + 0.004*"us" + 0.003*"also"
```

Visualization with pyldavis

```
1  import pyLDAvis
2  import pyLDAvis.gensim_models as gensimvis
3  # first estiate gensim model, then:
4  vis_data = gensimvis.prepare(lda_m1,ldacorpus_m1,id2word_m1)
5  pyLDAvis.display(vis_data)
```



Topic modelling

Choosing the best (or a good) topic model

Choosing the best (or a good) topic model

- There is no single best solution (e.g., do you want more coarse of fine-grained topics?)
- Non-deterministic
- Very sensitive to preprocessing choices
- Interplay of both metrics and (qualitative) interpretability

See for more elaborate guidance:

Maier, D., Waldherr, A., Miltner, P., Wiedemann, G., Niekler, A., Keinert, A., ... Adam, S. (2018). Applying LDA Topic Modeling in Communication Research: Toward a Valid and Reliable Methodology. *Communication Methods and Measures, 12*(2–3), 93–118. doi:10.1080/19312458.2018.1430754

Evaluation metrics

qualitative: human judgement

Observation and interpretation based: observe the top N words in your topic, and evaluate the quality of the coherence of the topic. Can you identify words that do not belong to a topic?

quantitative: coherence

- mean coherence of the whole model: attempts to quantify the interpretability
- coherence per topic: allows to get topics that are most likely to be coherently interpreted (`.top_topics()`)

So, how do we do this?

- Estimate multiple models, store the metrics for each model, and then compare them (numerically, or by plotting)
- Idea: We select some candidate models, and then look whether they can be interpreted.
- But what can we tune?

Choosing k: How many topics do we want?

- Typical values: $10 < k < 200$
- Too low: losing nuance, so broad it becomes meaningless
- Too high: picks up tiny pecularities instead of finding general patterns
- There is no inherent ordering of topics
- We can throw away or merge topics later, so if out of $k = 50$ topics 5 are not interpretable and a couple of others overlap, it still may be a good model

Choosing α: how sparse should the document-topic distribution θ be?

- The higher α, the more topics per document
- Default: $1/k$
- But: We can explicitly change it, or – really cool – even learn α from the data (`alpha = "auto"`)

```
mylda =LdaModel(corpus=tfidfcorpus[ldacorpus], id2word=id2word, num_topics=50, alpha='auto',
↪  passes=10)
```

Topic modelling

Using topic models

Using topic models

You got your model – what now?

1. Assign topic scores to documents
2. Label topics
3. Merge topics, throw away boilerplate topics and similar (manually, or aided by cluster analysis)
4. Compare topics between, e.g., outlets
5. or do some time-series analysis.

Example: Tsur, O., Calacci, D., & Lazer, D. (2015). A Frame of Mind: Using Statistical Models for Detection of Framing and Agenda Setting Campaigns. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing* (pp. 1629–1638).

Exercise for this week's lab session

- Work through the example notebook on LDA: https://github.com/uva-cw-ccs2/2324s2/blob/main/week03/exercise/topic-modelling.ipynb

Recommender Systems


You
Tube

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)

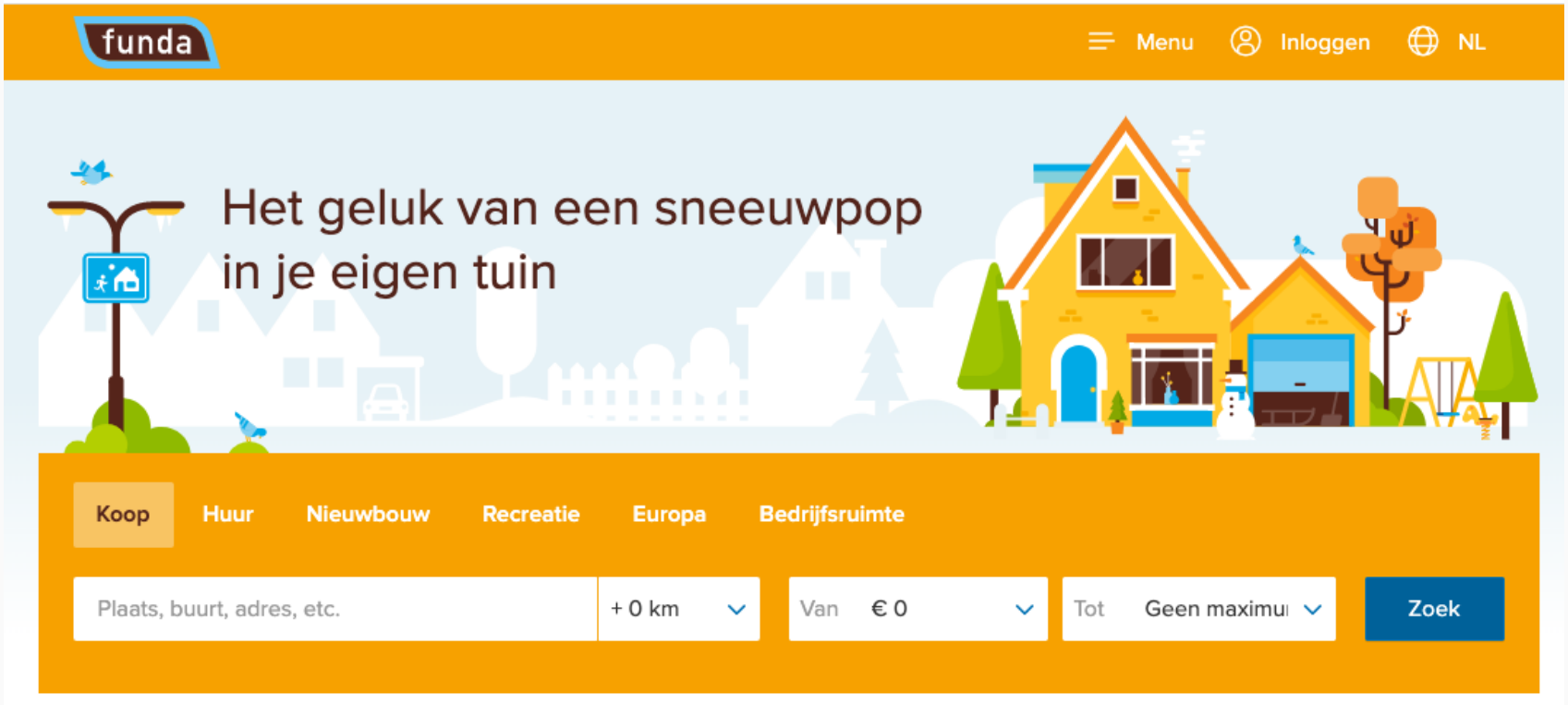
1. 'Basic' knowledge-base recommender systems
2. Content-based recommender systems
3. Collaborative recommenders

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
Menu
Inloggen
NL
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 maximu
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? $\rightarrow$ enter python's native input() function.

```
1 print("What is your favorite movie genre?")
2 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?

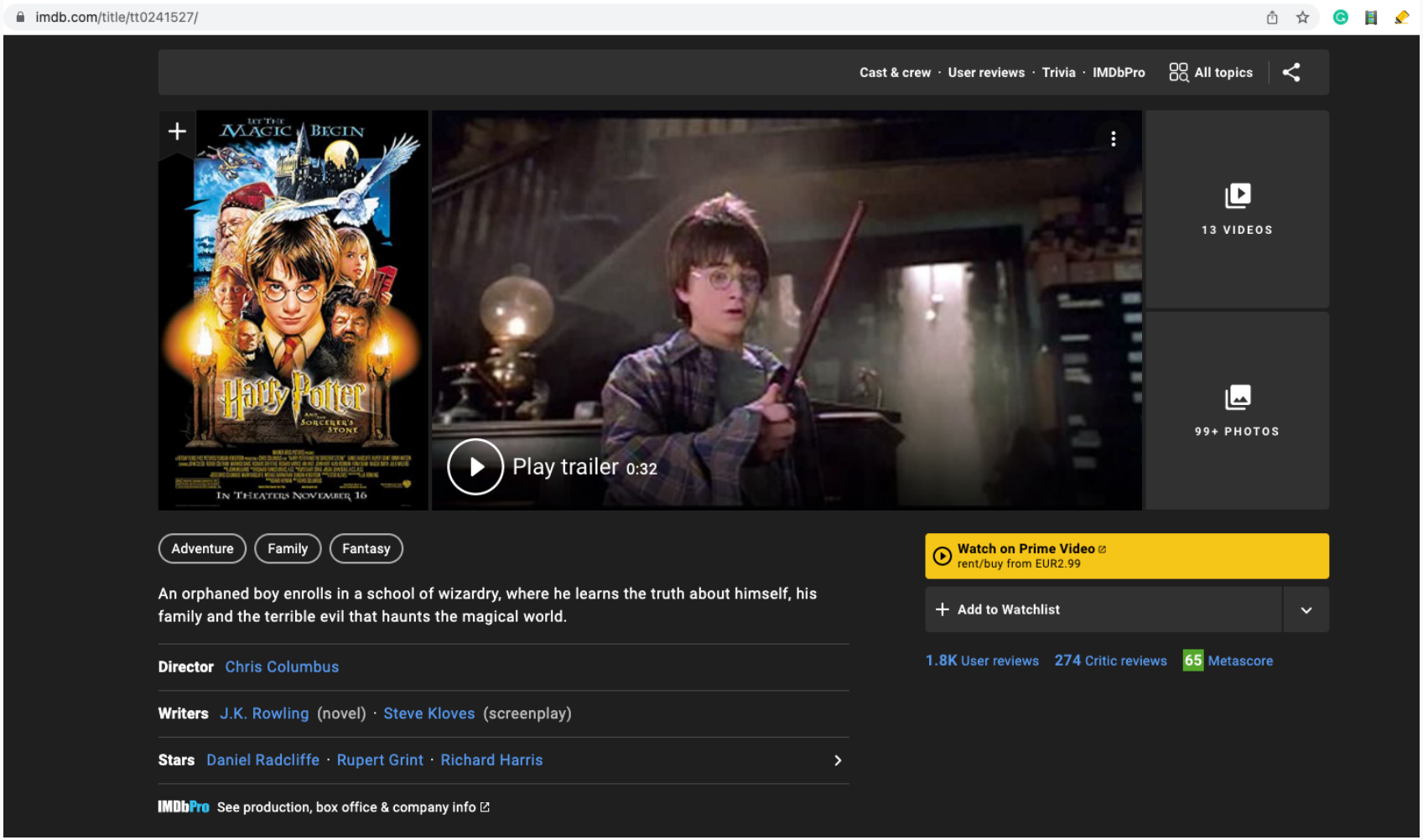
```
recommend_movies = movies.sort_values('vote_average',
↪  ascending=False)
```

Content-based RecSys

Content-based systems

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

Example of a content-based recsys



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

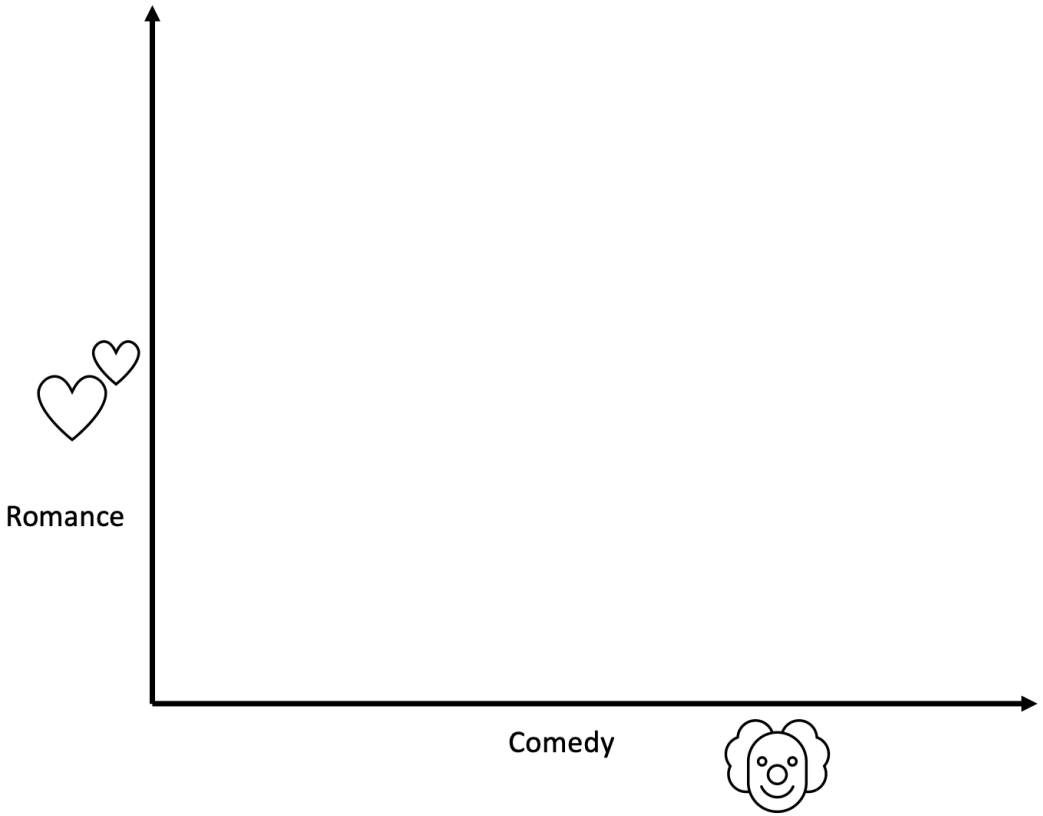
Use case: ImDb database

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

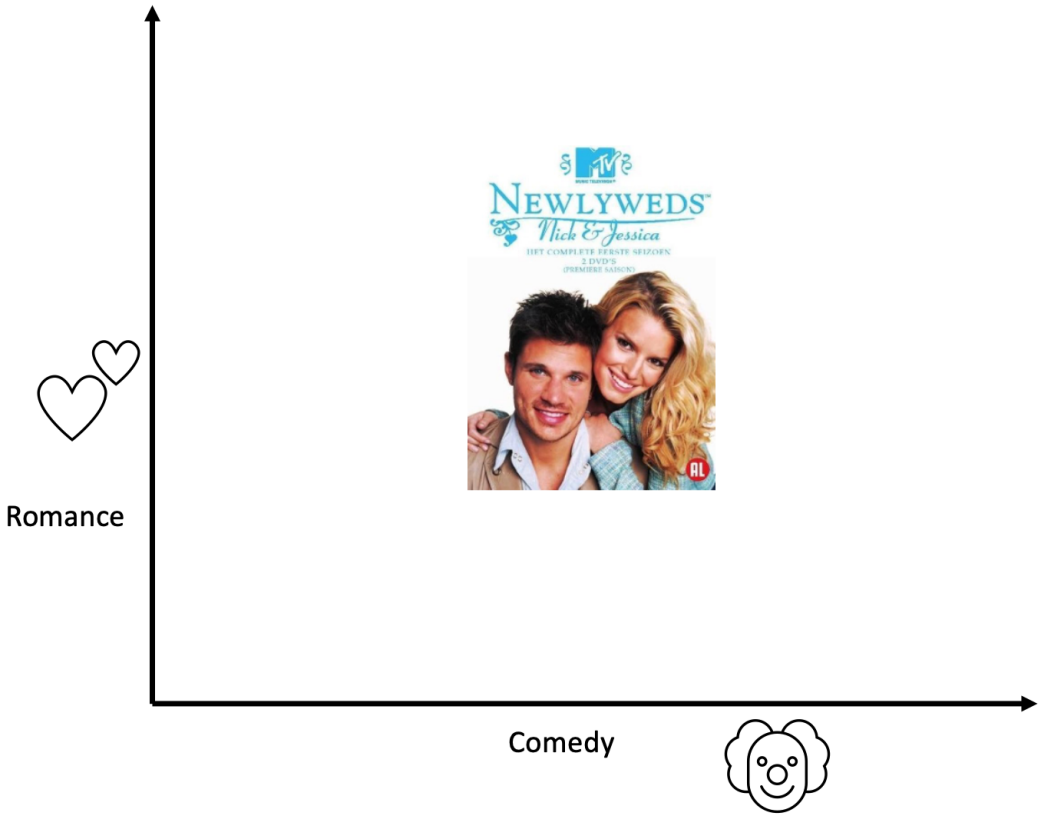
Are there attributes that you could use to estimate similarity in movies?

	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

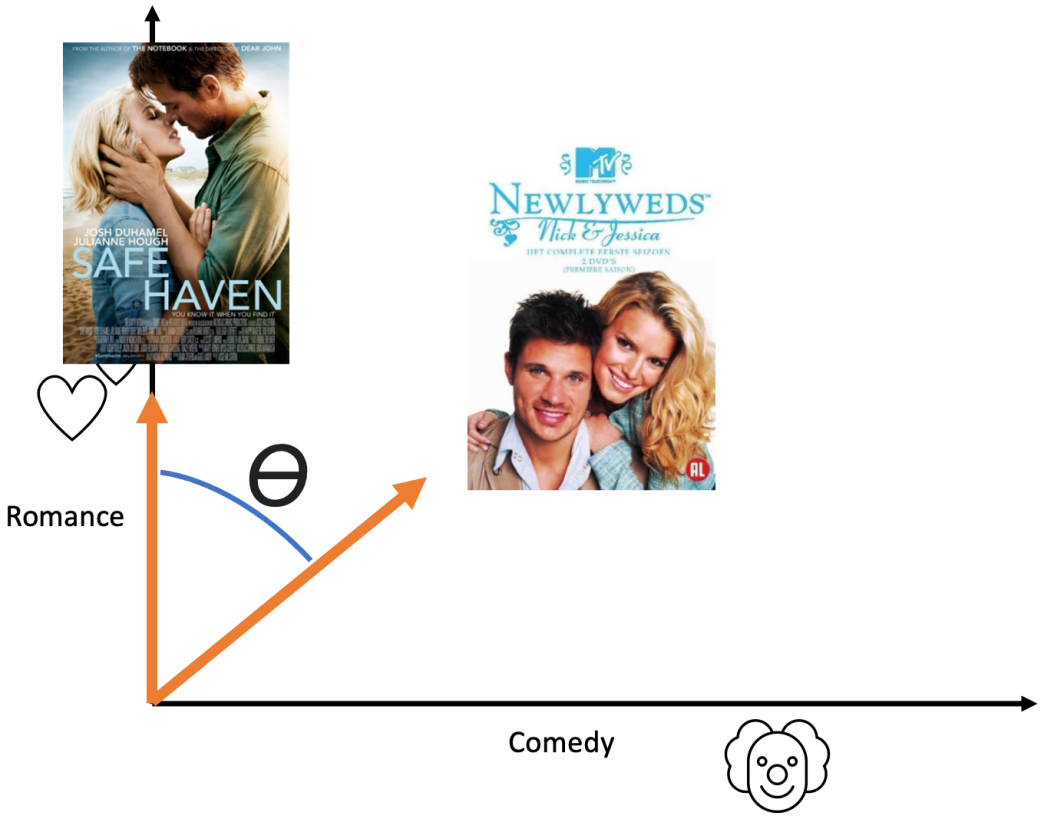


Similarity between movies

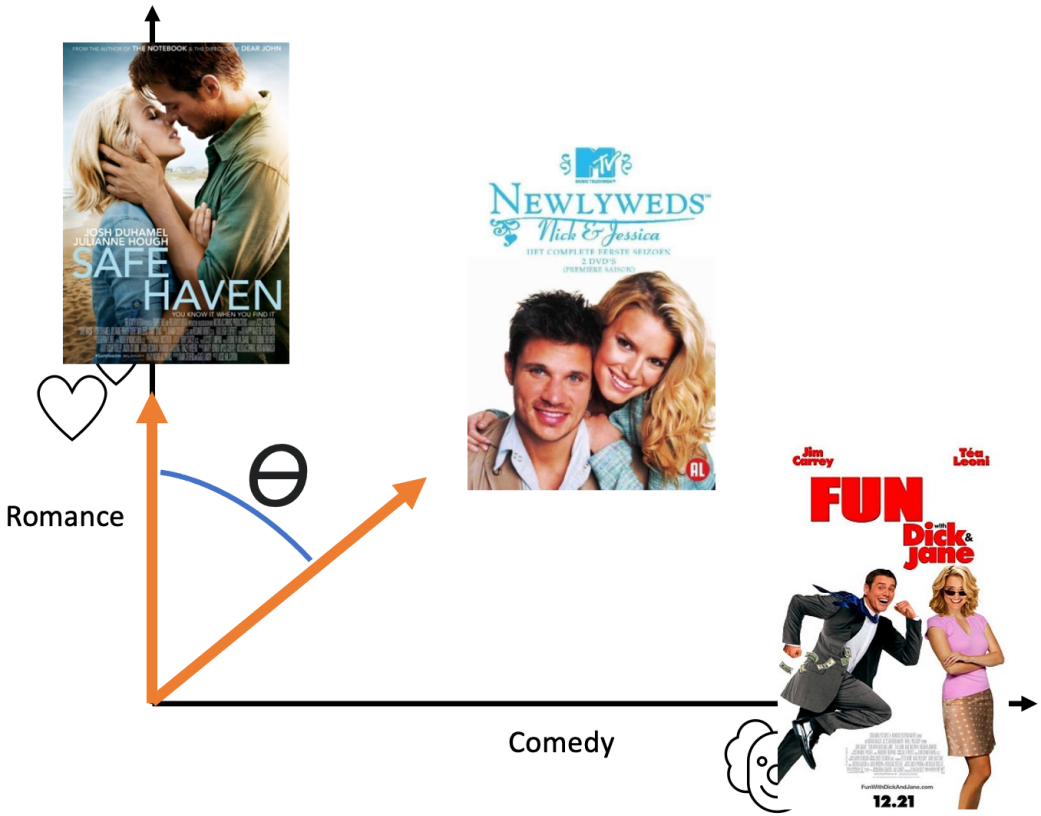


Similarity between movies

Similarity between movies



Similarity between movies



Let's put this in code!

First, create a create a toy dataset.

```
1 data = data.sample(6)
2 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 similiarity based on genres. Therefore, we need to vectorize this column.

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

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

Calculate similiarity

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

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

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

Cosine Similairity

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	genre	Avatar	Pirates of the Caribbean: At World's End	Spectre	The Dark Knight Rises	John Carter
		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_similarity to a df type of object, see here https://github.com/uva-cw-ccs2/2223s2/blob/master/week04/exercise/cosine_to_df.md.

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

Content-based RecSys

A more realistic use case

Example of a content-based recsys

ge	original_title	...	production_companies	production_countries	release_date	revenue	runtime	spoken_languages	status	tagline	vote_average	vote_count
en	avatar	...	[{"name": "Ingenious Film Partners", "id": 289...	[{"iso_3166_1": "US", "name": "United States o...	2009-12-10	2787965087	162.0	[{"iso_639_1": "en", "name": "English"}, {"iso...	Released	Enter the World of Pandora.	7.2	11800
en	pirates of the caribbean: at world's end	...	[{"name": "Walt Disney Pictures", "id": 2}, {"...	[{"iso_3166_1": "US", "name": "United States o...	2007-05-19	961000000	169.0	[{"iso_639_1": "en", "name": "English"}]	Released	At the end of the world, the adventure begins.	6.9	4500
en	spectre	...	[{"name": "Columbia Pictures", "id": 5}, {"nam...	[{"iso_3166_1": "GB", "name": "United Kingdom"...	2015-10-26	880674609	148.0	[{"iso_639_1": "fr", "name": "Fran\u00e7ais"},...	Released	A Plan No One Escapes	6.3	4466
en	the dark knight rises	...	[{"name": "Legendary Pictures", "id": 923}, {"...	[{"iso_3166_1": "US", "name": "United States o...	2012-07-16	1084939099	165.0	[{"iso_639_1": "en", "name": "English"}]	Released	The Legend Ends	7.6	9106
en	john carter	...	[{"name": "Walt Disney Pictures", "id": 2}]	[{"iso_3166_1": "US", "name": "United States o...	2012-03-07	284139100	132.0	[{"iso_639_1": "en", "name": "English"}]	Released	Lost in our world, found in another.	6.1	2124

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

```
data.head()
```

	movie_id	title	cast	crew	budget	genres	homepage	keywords	original_language	origin
0	19995	Avatar	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...	237000000	Action Adventure Fantasy Science Fiction	http://www.avatarmovie.com/	[{"id": 1463, "name": "culture clash"}, {"id":...	en	
1	285	Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847f800b579", "de...	300000000	Adventure Fantasy Action	http://disney.go.com/disneypictures/pirates/	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	en	pirate caribl wor
2	206647	Spectre	[{"cast_id": 1, "character": "James Bond", "cr...	[{"credit_id": "54805967c3a36829b5002c41", "de...	245000000	Action Adventure Crime	http://www.sonypictures.com/movies/spectre/	[{"id": 470, "name": "spy"}, {"id": 818, "name...	en	
3	49026	The Dark Knight Rises	[{"cast_id": 2, "character": "Bruce Wayne / Ba...	[{"credit_id": "52fe4781c3a36847f81398c3", "de...	250000000	Action Crime Drama Thriller	http://www.thedarkknightrises.com/	[{"id": 849, "name": "dc comics"}, {"id": 853,...	en	1 knig

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.

```
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:

```
indices = pd.Series(data.index, index = data['original_title'])
index = indices['the dark knight rises']
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 similair 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]))
```

```
1 [(0, 0.023313489832942038),
2 (1, 0.003991972883233364),
3 (2, 0.007801739082093903), ...
```

Next, we can simple sort the cosine scores:

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

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
```

```
1 [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

Content-based RecSys

Building blocks of content-based RecSys

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 wil 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 suprising recommendations.
- Very succesful models in industry settings

Types of collaborative systems

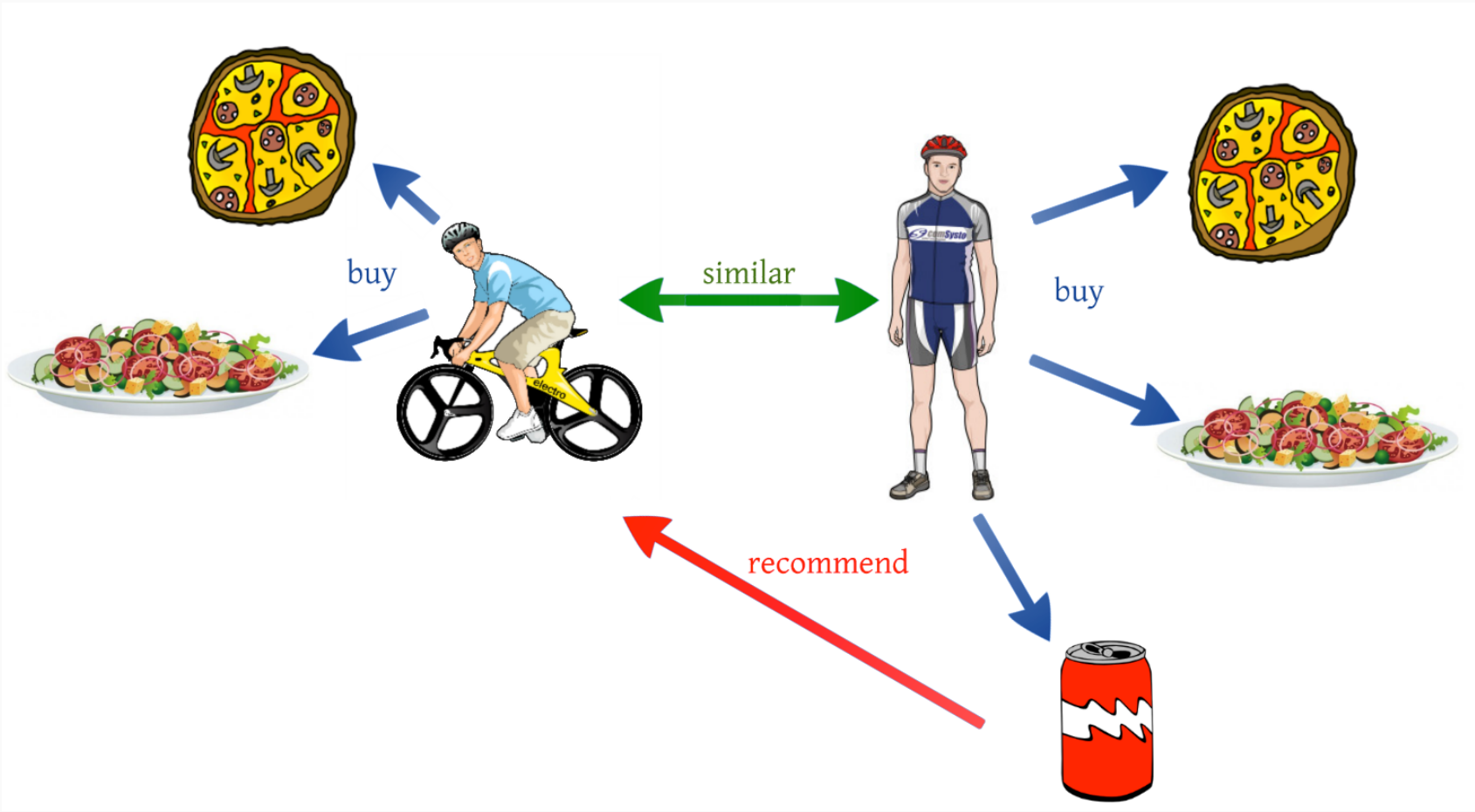
1. User-based collaborative filtering
2. Item-based collaborative filtering

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] Source: https://medium.com/@akhilesh3091999/recommender-system-bb032b16ab67

Item-based filtering

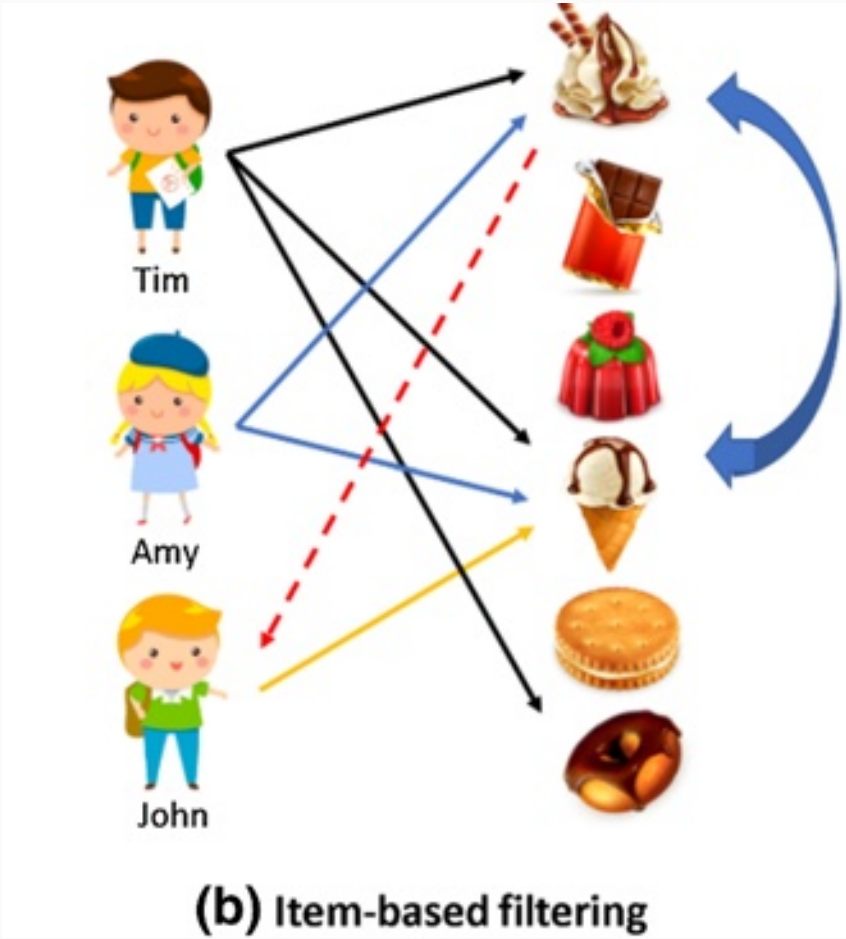
Similar items...

- If two items is rated similarly by a group of people
- ...these product 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



(b) Item-based filtering

[2]

[2] Source: https://predictivehacks.com/item-based-collaborative-filtering-in-python/

Example: Amazon

Reflection on the role of recommender systems in society

Food for thought

- Consequences of recommender systems for democratic values in society

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.

Have fun!

References i

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

Maier, D., Waldherr, A., Miltner, P., Wiedemann, G., Niekler, A., Keinert, A., Pfetsch, B., Heyer, G., Reber, U., Häussler, T., Schmid-Petri, H., & Adam, S. (2018). Applying LDA Topic Modeling in Communication Research: Toward a Valid and Reliable Methodology. *Communication Methods and Measures*, *12*(2-3), 93–118. https://doi.org/10.1080/19312458.2018.1430754

References ii

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

Rehurek, R., & Sojka, P. (2010). Software framework for topic modelling with large corpora. *IN PROCEEDINGS OF THE LREC 2010 WORKSHOP ON NEW CHALLENGES FOR NLP FRAMEWORKS*, 45–50.

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