Computational Communication Science 2
Week 3 - Lecture
» (Soft) cosine and recommender systems«

Anne Kroon

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

April 14, 2025

Digital Society Minor, University of Amsterdam

Today

Everything clear from last weeks?

Recap: Week 1+2

What We Covered in the Last Weeks

Text Preprocessing

- **Tokenization**: Splitting text into words or phrases.
- **Stopword Removal**: Filtering out common, low-information words.
- **N-grams**: Creating combinations of words (e.g., unigrams, bigrams, trigrams, collocations).
- **Stemming/Lemmatization**: Reducing words to their root or base form.

Text representations (Numerical form)

- **Count Vectorization**: Representing text as word occurrence counts.
- **TF-IDF Vectorization**: Weighs word importance relative to the document and corpus.
- **Embeddings**: Vector representations capturing semantic meaning (spaCy).

What We Covered in the Last Weeks

Top down and bottom up approaches

- We have discussed top-down and bottom-up approaches....
- ...today we'll talk about a *bottom-up* approach:
  - cosine similiarity
- ...second half of the course will focus on *top-down*
  - machine learning (vectorizers + pre-processing remain important!)

The Bigger Picture

Comparing CCS-1 and CCS-2

- **CCS-1** focused on the *basics of programming* — like learning how to drive.
- **CCS-2** focuses on *applying computational techniques to analyze textual data* — more like learning how the engine works.
- This course emphasizes conceptual understanding and methodological application.
  - You'll be tested on your ability to apply concepts — not on writing complex code from scratch.

The Bigger Picture

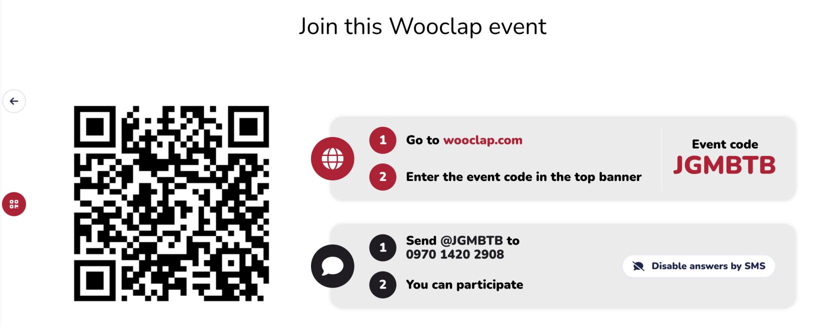
Comparing CCS-1 and CCS-2

- **CCS-2** introduces more abstract concepts. Yes, it can be challenging — but you **can** do hard things! Mastering these ideas is empowering.
- Topics like *vectorization*, *cosine similarity*, and *machine learning* aren't just for this course — they're valuable, in-demand skills that look great on your resume.
- If you're having trouble, please reach out — we're here to support you!
- **Have feedback or want a consultation?** Tell us how we can support you better: `https://forms.office.com/e/7fzzTvKGyT?origin=lprLink`

Recap: Test Your Understanding of Vectors

Did You Get It?

- Participate here: https://app.wooclap.com/JGMBTB?from=event-page



Document-Term Matrix Comparison (Count, TF-IDF, Embedding)

Count Vectorizer

Doc	cat	sat	dog
D1	1	1	0
D2	1	0	1
D3	0	1	1

Raw word frequencies

TF-IDF Vectorizer

Doc	cat	sat	dog
D1	0.58	0.58	0.00
D2	0.48	0.00	0.66
D3	0.00	0.48	0.66

Weights rare/important words

Embedding (spaCy avg.)

Doc	dim1	dim2	dim3
D1	0.31	-0.04	0.88
D2	0.27	-0.12	0.95
D3	0.26	-0.01	0.83

Dense semantic vectors (truncated)

Cosine similarity

Why Should We Care?

Cosine Similarity in Action

- Ever wonder how Spotify knows your next favorite song?
- Or why Netflix keeps recommending crime thrillers?
- Behind the scenes, they're comparing items as vectors.
- Cosine similarity tells us how "close" two items are in meaning or content.

Key Idea

Two documents (or songs, users, products) are similar if their vector directions are close — even if their values (lengths) differ.

Applications of Cosine Similarity

Where It's Used

- **In industry:**
  - Search engines (e.g., ranking relevant results)
  - Recommendation systems (e.g., suggesting similar content)
- **In academia:**
  - *Linguistic alignment in communication* (Brinberg & Ram, 2021)
  - *Overlap between political speech and public opinion* (Hager & Hilbig, 2020)

From Vectorization to Similarity

Good news: You already know (almost) everything!

- You've worked with vectors
- You already know how to **vectorize** your data using `CountVectorizer` and `TfidfVectorizer`
- Now, you can use those vectors to calculate **cosine similarity** — it's just one more step!

What This Means

Once your text is in vector form, you can compare:

- How similar two documents are
- Whether two users talk alike
- Which sentences match a query best

Mathematical Representation

$$\text{similarity} = \cos(\theta) = \frac{\mathrm{A} \cdot \mathrm{B}}{\|\mathrm{A}\| \|\mathrm{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

- Measures cosine of the angle between vectors.
- 0 (orthogonal, dissimilar), 1 (identical, similar).

Interpreting Cosine Similarity Values

What the Numbers Tell You

Cosine similarity values range from **0 to 1**:

- **1.0** → Texts are *identical in direction* (highly similar).
- **0.8 – 1.0** → Very similar content or style.
- **0.5 – 0.8** → Some overlap in meaning, but not identical.
- **0.0 – 0.5** → Low similarity; texts likely on different topics.
- **0.0** → No overlap in direction (completely dissimilar).

Tip

Cosine similarity is all about the *direction* of vectors — so it's great when document length doesn't matter, but shared emphasis (on certain words or ideas) does.

Cosine Similarity in Python

You've already seen how to vectorize text — now let's add one more line to compute similarity!

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd
# Sample documents
documents = [
    "When I eat breakfast, I usually drink some tea",
    "I like my tea with my breakfast",
    "She likes cereal and coffee"
]
# Vectorize the text (bag-of-words)
vec = CountVectorizer(stop_words='english') # OR use TfidfVectorizer()
count_matrix = vec.fit_transform(documents)
# Compute cosine similarity
cos_sim = cosine_similarity(count_matrix) --> Only this line is new :-)
# Display as DataFrame for better readability
print(pd.DataFrame(cos_sim))
```

Beyond Cosine: Introducing Soft Cosine Similarity

Limitations of Cosine using `CountVectorizer` and `TfidfVectorizer`.

- Only works with exact word matches (e.g. "car" ≈ "automobile").
- Doesn't capture deeper **semantic relationships**.

Soft Cosine Similarity

- Uses **word embeddings** (spaCy) to measure similarity even with different words.
- Captures **synonyms**, related terms, and contextual meaning (Sidorov et al., 2014).

Cosine vs Soft Cosine: Side-by-Side Comparison

Comparing Two Sentences

Sentence A: "I drove my car to work."
Sentence B: "I drove my automobile to work."

Regular Cosine Similarity

- Method: `CountVectorizer` or `TfidfVectorizer`
- Only exact word matches
- **Score: 0.67**
- Misses that "car" and "automobile" are related

Soft Cosine Similarity (Embeddings)

- Method: Word Embeddings (spaCy)
- Captures semantic relationships
- **Score: 0.95**
- Understands that "car" ≈ "automobile"

Key Takeaway

Soft cosine captures **meaning**, not just word overlap — more powerful for nuanced language tasks.

Let's put this into practice!

Check out the walkthrough here:
https://github.com/uva-cw-ccs2/2425s2/blob/main/week03/exercise-lecture/cosine_similarity_WALKTHROUGH.ipynb

RecSys

Congratulations! You now have everything you need to build a recommender system.

Recommender Systems in Communication Science

New Research Questions

1. Political communication and journalism. E.g., crafting personalized news diets. However, this may impact the diversity of news diets and democracy (Locherbach & Trilling, 2018; Möller et al., 2018)
2. Organizational and corporate communication. E.g., applications in hiring and recruitment.
3. Persuasive communication. E.g., recommendation algorithms for tailored health interventions (Kim et al., 2019)
4. Entertainment communication. E.g., movie recommenders.

Recommender Systems

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

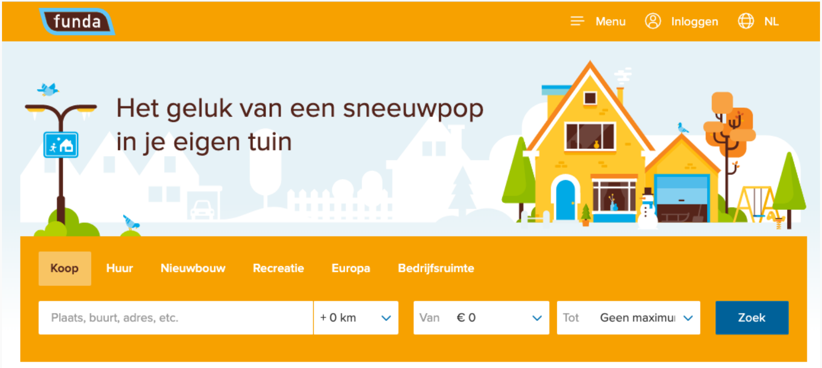
1. 'Basic' knowledge-base recommender systems
2. Content-based recommender systems
3. Collaborative recommenders (not part of this course)

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? → enter python's native input() function.

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

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

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?

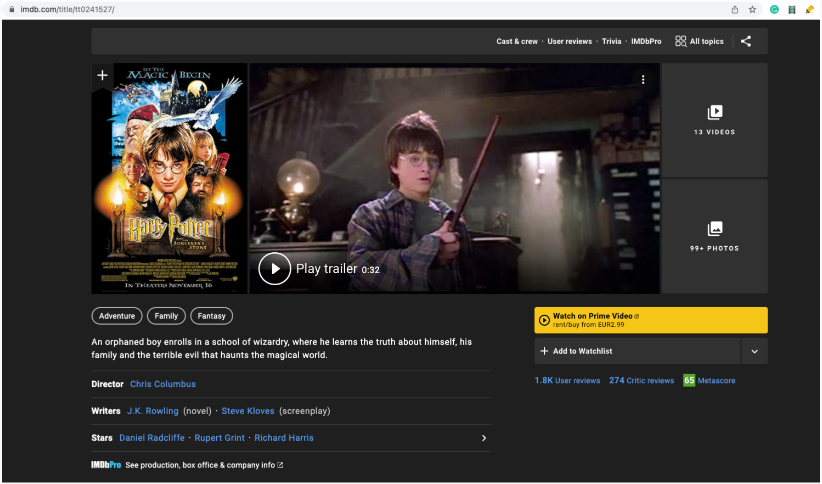
```
recommend_movies = movies.sort_values('vote_average',
    ascending=False)
```

Content-based RecSys

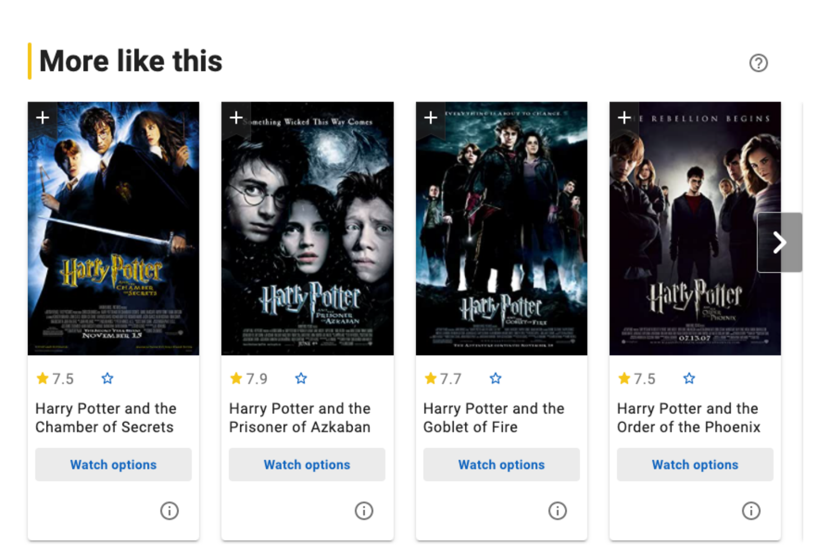
Content-based systems

- 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



Content-based RecSys

Building blocks of content-based RecSys

Feature selection and preprocessing

- Feature engineering is essential. What attributes or qualities do we want to include? In other words, which columns will you select and combine? (more on this tomorrow)
- Preprocessing your data (e.g., removing stop words, stemming) is important for improving cosine similarity with CountVectorizer and TF-IDF. However, for soft cosine (using embeddings), preprocessing is not required.

Now how can we identify similar items?

1. Cosine similarity using text transformed with a CountVectorizer
2. Cosine similarity using text transformed with a TfidfVectorizer
3. Soft cosine similarity using word embeddings from an embedding model

Benefits

- Content-based recommender systems are efficient and can provide highly personalized recommendations based on individual preferences.
- They are often integral to more complex recommender systems that combine deep learning and supervised learning techniques.

You now know how to implement these!

- You've learned how to vectorize text and compute similarities — the foundation of content-based recommendations.
- With this knowledge, you can easily build effective, data-driven recommendation systems!

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.

Wrap up

What We Covered in Part 1 – CCS 2: All ingredients of Content-based RecSys

Text Preprocessing: Tokenization, Stopword Removal, N-grams, Stemming/Lemmatization

Text Representations: Count Vectorization, TF-IDF, Embeddings

Cosine and Soft Cosine Similarity

What's Next?

Apply Knowledge in
Group Assignment

↓

Continue with Saurabh Khanna
after the Education-Free Week

Thank You!

Questions or feedback?

a.c.kroon@uva.nl

References i

References

Brinberg, M., & Ram, N. (2021). **Do new romantic couples use more similar language over time? Evidence from intensive longitudinal text messages.** *Journal of Communication*, *71*(3), 454–477. https://doi.org/10.1093/joc/jqab012

Hager, A., & Hilbig, H. (2020). **Does public opinion affect political speech?** *American Journal of Political Science*, *64*(4), 921–937. https://doi.org/10.1111/ajps.12516

References ii

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

References iii

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

Sidorov, G., Gelbukh, A., Gómez-Adorno, H., & Pinto, D. (2014).**Soft similarity and soft cosine measure: Similarity of features in vector space model.** *Computacion y Sistemas*, *18*(3), 491–504.
https://doi.org/10.13053/CyS-18-3-2043

References iv

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