



# Simple book recommender

Tomáš Novák

tomas2211@post.cz

[github.com/tomas2211/book\\_recommender](https://github.com/tomas2211/book_recommender)

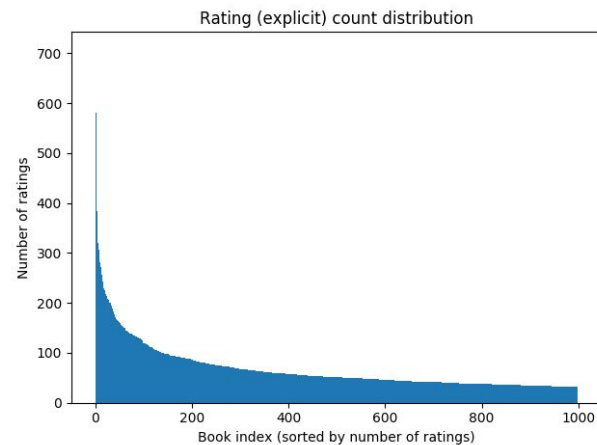
# Outline



- Dataset + Task
- Approaches
  - Simple baseline approach
  - Graph-based approaches - close vertices, node2vec
  - K nearest neighbours
- Evaluation
  - NDCG
  - Qualitative evaluation
- Deployment

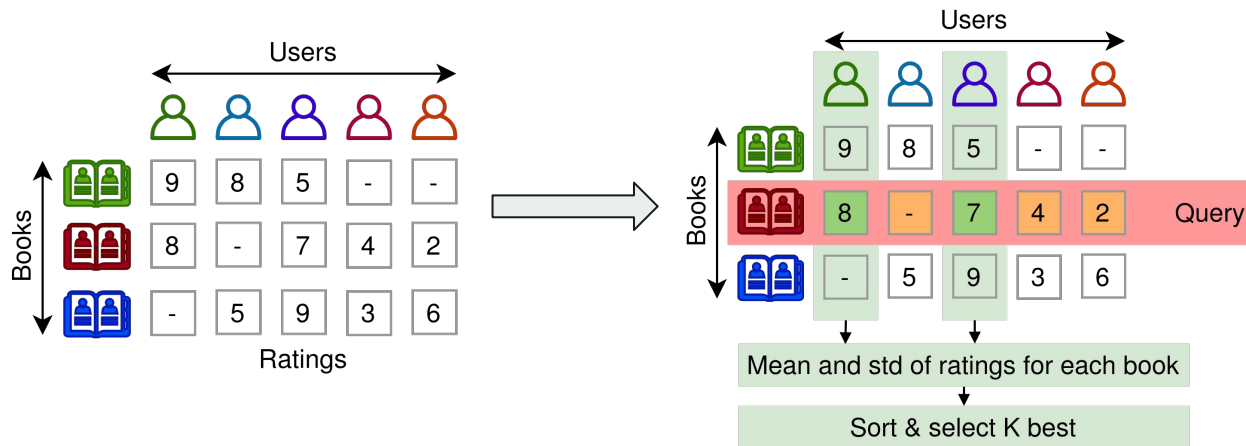
# Dataset + Task

- Data: User ratings
  - Columns - user-id, book-id, rating
  - Ratings - Implicit (0), explicit (1-10)
    - Considering only explicit ratings
  - Many books with small number of ratings
    - Considering only books with >10 ratings
    - → 4963 books
- Task
  - Input: Query (book ISBN)
  - Output: Top K recommended books



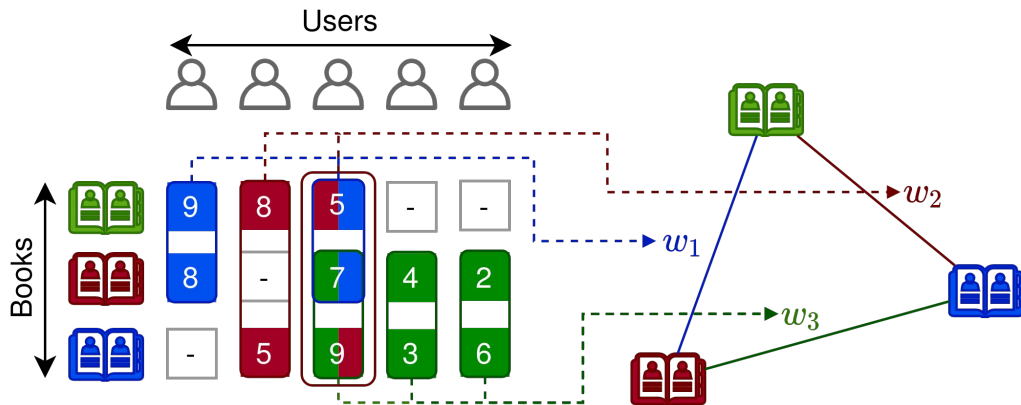
# Baseline approach

- Idea: model conditional prob. distributions of ratings for each book
- $P(\text{rating of book} \mid \text{liked query}) \approx N(\mu, \sigma^2)$
- Liked the query: rating > 5 (fixed threshold)
- Maximize the minimum expected rating on some level of confidence i.e.  $\max(\mu - S \cdot \sigma)$



# Graph-based approach

- Idea: model relations between books in a graph (vertices = books, edges = relations, weighted by distance/closeness)
- Possibility to explore deeper relations, find niches
- Weights calculated by considering each pair of user's ratings



# Graph-based approach

- Edge weights
  - User's  $u$  rating of book  $k$ :  $r_u(k)$
  - **Co-rating** of books  $k$  and  $l$ :  $r_u(k)r_u(l)$
  - Mean and std over users  $u$  that rated  $k$  and  $l$ , normalized:

$$\mu(k, l) = \frac{1}{N} \sum_{u: r_u(k) > 0, r_u(l) > 0} \frac{r_u(k)r_u(l)}{100}$$

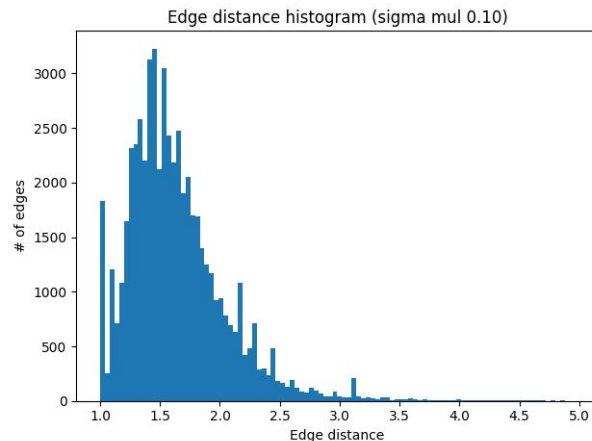
$$\sigma(k, l) = \sqrt{\frac{1}{N} \sum_{u: r_u(k) > 0, r_u(l) > 0} \left( \frac{r_u(k)r_u(l)}{100} - \mu(k, l) \right)^2}$$

(+ disregard edges with less than 3 co-ratings, ...)

# Close vertices - Graph-based approach

- Search around query vertex - limit depth
- Sort other vertices by distance (path length) to query vertex
- Needs “distance measure” - proportional to dissimilarity
  - Inverse mean co-rating
  - “Standard deviation trick” with parameter  $S$

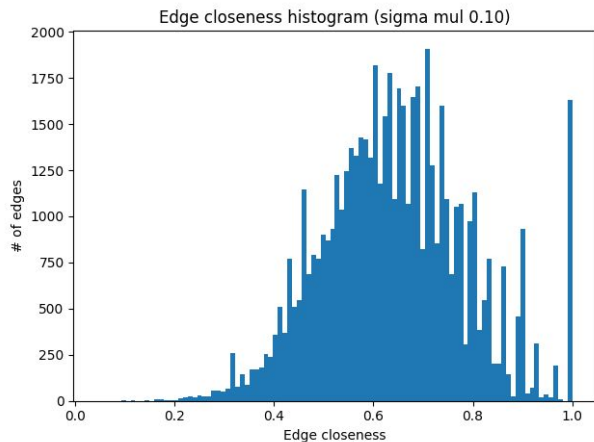
$$d(k, l) = \frac{1}{(\mu(k, l) - S\sigma(k, l))}$$



# Node2Vec - Graph-based approach

- Create vector representations (embeddings) for nodes in graph
- Based on Word2Vec (sequences of nodes by random walks)
- Compare vectors with query embedding (cosine distance)
- Edges need “closeness measure”
  - Mean co-rating normalized to range 0 - 1
  - “Standard deviation trick”

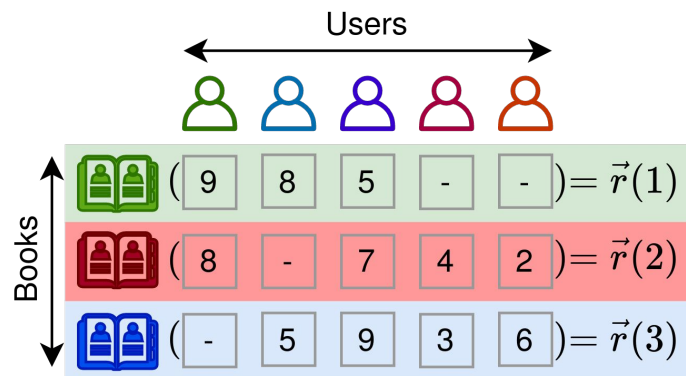
$$c(k, l) = (\mu(k, l) - S\sigma(k, l))$$





# k-nearest neighbours

- Conventional approach
- Represent books as vectors of user ratings
- Cosine similarity with query book
  - Similar to co-ratings, but normalized differently
- Measures similarity instead of probability of liking a book
  - If one users dislikes two books - makes them more similar
- Potential problem on less-rated books - cannot step 'over a user'



$$sim(k, l) = \frac{\sum_u r_u(k) r_u(l)}{\sqrt{\sum_u r_u^2(k)} \sqrt{\sum_u r_u^2(l)}}$$

# Quantitative evaluation



- Randomly selected test users (excluded from training)
  - 40 users with more than 50 explicit ratings
- Query each 'liked' book of test user (>5 rating)
- Response - top 10 books
- Compute NDCG on response
  - Sum user ratings of responses, weighted by place
  - Unrated books - rating = 0
  - Normalize by maximum possible score for the user

$$\frac{1}{\log_2(place + 1)}$$

# Quantitative evaluation



- Overall, low scores
  - Rating sparsity?
  - Single liked book may not bring much information about the user
  - Maximal attainable NDCG?
- Better than a random selection
  - Probability of randomly hitting a rated book is 1.6%
- Graph - close nodes
  - Better selection of rated nodes
  - Worse cumulative gain - distribution of distances to narrow?

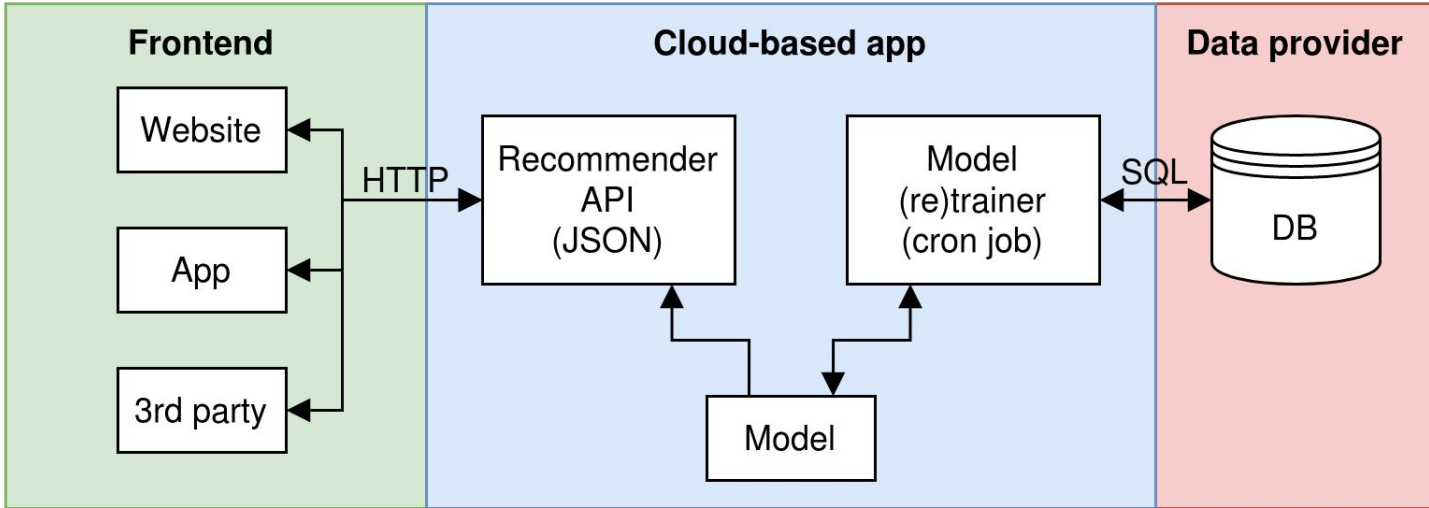
Approach	NDCG	Rated books in responses [%]
Baseline	0.092	10.7
Graph - close vertices	0.097	16.6
Graph - node2vec	0.102	12.1
kNN	0.118	12.6

# Qualitative evaluation

- Mostly relevant or at least not irritating suggestions, but with exceptions (selections from top 5)

Approach	J. R. R. Tolkien: The Hobbit	Chaim Potok: The Chosen	Douglas Adams: Hitchhiker's Guide to the Galaxy
Baseline	J. K. Rowling: Harry Potter and the Sorcerer's Stone	Margaret Atwood: The Handmaid's Tale [dystopia]	Jon Krakauer: Into Thin Air : A Personal Account of the Mt. Everest Disaster [non-fiction]
Graph - close vertices	J. K. Rowling: Harry Potter and the Sorcerer's Stone	John Le Carre: The Tailor of Panama [spy novel]	Dan Brown: The Da Vinci Code
Graph - node2vec	Helen Fielding: Das Tagebuch Der Bridget Jones	John Le Carre: The Tailor of Panama [spy novel]	Douglas Adams: The Restaurant at the End of the Universe
kNN	Martha Sacks: Menopaws: The Silent Meow	Chaim Potok: My Name Is Asher Lev	Douglas Adams: The Restaurant at the End of the Universe.

# Productionalization



# Google Cloud - App Engine

- JSON API in Flask, static model (kNN) - see *gcp\_app* folder in repo
- Matches name against DB & evaluates the model

[https://abiding-ripple-272918.ew.r.appspot.com/query?name=\[book-name\]](https://abiding-ripple-272918.ew.r.appspot.com/query?name=[book-name])

The screenshot shows a REST client interface with a GET request to the endpoint `https://abiding-ripple-272918.ew.r.appspot.com/query?name=The Fellowship of the Ring (The Lord of the Rings, Part 1)`. The response is a JSON object with the following structure:

```
{
  "isbns": [
    "0345339738",
    "0345339711",
    "0345339681",
    "051508901X",
    "034538475X"
  ],
  "names": [
    "J.R.R. TOLKIEN: The Return of the King (The Lord of the Rings, Part 3)",
    "J.R.R. TOLKIEN: The Two Towers (The Lord of the Rings, Part 2)",
    "J.R.R. TOLKIEN: The Hobbit : The Enchanting Prelude to The Lord of the Rings",
    "Lavyrie Spencer: The Gamble",
    "Anne Rice: The Tale of the Body Thief (Vampire Chronicles (Paperback))"
  ],
  "query_name": "J.R.R. TOLKIEN: The Fellowship of the Ring (The Lord of the Rings, Part 1)",
  "scores": [
    0.6179244415879612,
    0.6259065296773095,
    0.783084521631701,
    0.9170334707177676,
    0.9193296959857358
  ]
}
```

# Demo

The screenshot displays a GraphQL IDE interface. On the left, a sidebar shows a 'Collections' panel with a tree view containing 'DB BOOK' and 'Book recommendations'. The main workspace is titled 'My Workspace' and shows a query editor with the following query:

```
GET {
  http://localhost:3000/graphql
  query: query {
    books {
      title
      author
      year
    }
  }
}
```

Below the query editor, a table displays the results of the query:

id	title	author
1	Harry Potter and the Sorcerer's Stone	J.K. Rowling
2	Harry Potter and the Chamber of Secrets	J.K. Rowling
3	Harry Potter and the Prisoner of Azkaban	J.K. Rowling
4	Harry Potter and the Goblet of Fire	J.K. Rowling
5	Harry Potter and the Order of the Phoenix	J.K. Rowling
6	Harry Potter and the Half-Blood Prince	J.K. Rowling
7	Harry Potter and the Deathly Hallows	J.K. Rowling

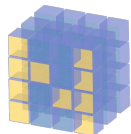
At the bottom, a 'Query' panel shows the JSON response of the query:

```
{
  "data": {
    "books": [
      {
        "id": "1",
        "title": "Harry Potter and the Sorcerer's Stone",
        "author": "J.K. Rowling"
      },
      {
        "id": "2",
        "title": "Harry Potter and the Chamber of Secrets",
        "author": "J.K. Rowling"
      },
      {
        "id": "3",
        "title": "Harry Potter and the Prisoner of Azkaban",
        "author": "J.K. Rowling"
      },
      {
        "id": "4",
        "title": "Harry Potter and the Goblet of Fire",
        "author": "J.K. Rowling"
      },
      {
        "id": "5",
        "title": "Harry Potter and the Order of the Phoenix",
        "author": "J.K. Rowling"
      },
      {
        "id": "6",
        "title": "Harry Potter and the Half-Blood Prince",
        "author": "J.K. Rowling"
      },
      {
        "id": "7",
        "title": "Harry Potter and the Deathly Hallows",
        "author": "J.K. Rowling"
      }
    ]
  }
}
```

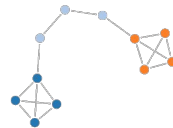
# Conclusions

- Designed several approaches to book recommendation
  - Baseline
  - Graph-based - close vertices, node2vec
  - kNN
- Evaluation - NDCG, qualitative
- Productionalization - design
- Prototype deployment to GCP - App Engine

Technologies: Python, Pandas, Numpy, Scikit-learn, NetworkX, Flask  
Google cloud platform - App Engine



NumPy



NetworkX



Flask  
web development,  
one drop at a time