Book recommendation search engine

Lukáš Dobiš

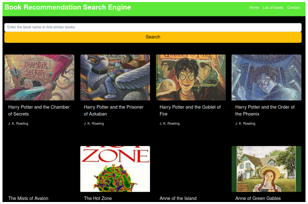
dobis.lks@gmail.com

August, 2023

Solution preview

A product capable of delivering recommendations of similar books, based on submitted book query.

Examples below are from deployed prototype solution.


Book Recommendation Search Engine
Home
List of books
Contact
Enter the book name to find similar books.
Search

Harry Potter and the Chamber of Secrets
J. K. Rowling

Harry Potter and the Prisoner of Azkaban
J. K. Rowling

Harry Potter and the Goblet of Fire
J. K. Rowling

Harry Potter and the Order of the Phoenix
J. K. Rowling

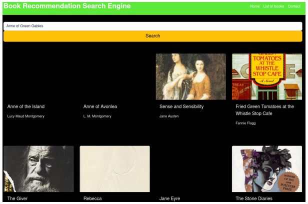
The Mists of Avalon

The Hot Zone

Anne of the Island

Anne of Green Gables

Query: Harry Potter



Anne of the Island
Lucy Maud Montgomery

Anne of Avonlea
L. M. Montgomery

Sense and Sensibility
Jane Austen

Fried Green Tomatoes at the Whistle Stop Cafe
Fannie Flagg

The Giver

Rebecca

Jane Eyre

The Stone Diaries

Query: 1984

Data

Book-Crossing (BX) Dataset - small dataset from book website. Limited by ratings up to year 2004, size and users demographic. For research use only.

Other data sources

- *Goodreads*
- *Amazon preview data*
- *OpenLibrary*

	Ratings	Books	Users	Raw size
BX	1 149 780	271 379	278 858	115 MB
Goodreads	104 551 549	2 360 655	876 145	28.1 GB
Amazon	51 311 621	2 935 525	-	35.1 GB

Ziegler, C.N., McNee, S.M., Konstan, J.A. and Lausen, G., 2005, May. Improving recommendation lists through topic diversification. In Proceedings of the 14th international conference on World Wide Web (pp. 22-32). http://www2.informatik.uni-freiburg.de/ cziegler/BX/
Justifying recommendations using distantly-labeled reviews and fined-grained aspects Jianmo Ni, Jiacheng Li, Julian McAuley Empirical Methods in Natural Language Processing (EMNLP), 2019: https://cseweb.ucsd.edu/ jmcauley/datasets/amazon$_v$2/
Wan, M. and McAuley, J., 2018, September. Item recommendation on monotonic behavior chains. In Proceedings of the 12th ACM conference on recommender systems (pp. 86-94).: https://mengtingwan.github.io/data/goodreads.html

Data exploration of BX data

- books.csv - multiple editions for same Book, inconsistent naming for same Author, unimportant Publisher information, some missing values for Year-Of-Publication
- ratings.csv - half of ratings is either zero or larger than 5 indicating users rating behaviour
- users.csv - information either missing (Age) or not useful (1/3 of users have no ratings) or biased (70% of users is from USA)

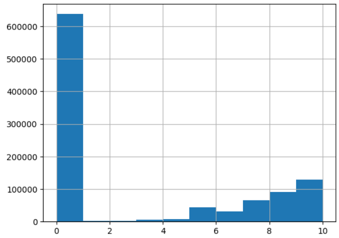


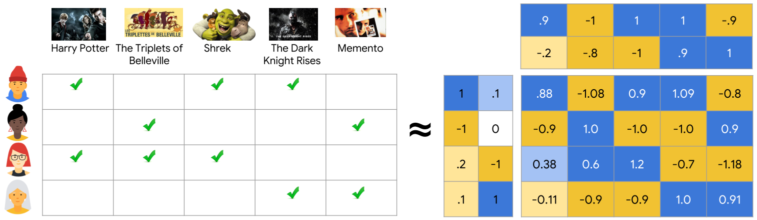
Figure: Histogram of book rating values

Data cleaning of BX data

- books.csv - removing suffixes from book title, standardizing authors initials spacing, assigning unique book ID to each unique book title, leaving only lowest year of publication year
- ratings.csv - grouping different ISBN codes under one Book ID to identify ratings for specific book, removing ratings from users with low count of ratings, removing ratings from books with low count of ratings.
- users.csv - not used
- Remainder of books and ratings in dataframes/tables have after filtering additionally assigned new IDs: Book, Author, Year of publication. These IDs are used to index embeddings.

Collaborative filtering

Recognizing patterns in item to user behaviour to recommend an item to user A based on the interests of a similar user B. User-Item interactions can be viewed in User-Item matrix. User and book embedding vectors can be learned through approximation of this matrix.



	Harry Potter	The Triplets of Belleville	Shrek	The Dark Knight Rises	Memento
User 1	✓		✓	✓	
User 2		✓			✓
User 3	✓	✓	✓		
User 4				✓	✓

$$\approx \begin{array}{cc|ccccc} & & .9 & -1 & 1 & 1 & -.9 \\ & & -.2 & -.8 & -1 & .9 & 1 \\ \hline 1 & .1 & .88 & -1.08 & 0.9 & 1.09 & -0.8 \\ -1 & 0 & -0.9 & 1.0 & -1.0 & -1.0 & 0.9 \\ .2 & -1 & 0.38 & 0.6 & 1.2 & -0.7 & -1.18 \\ .1 & 1 & -0.11 & -0.9 & -0.9 & 1.0 & 0.91 \end{array}$$

Figure: User - Item matrix and its approximation by Matrix Factorization

Image source: https://developers.google.com/machine-learning/recommendation/collaborative/matrix

Retrieval

- cosine similarity - (*Item C*) - similarity as smallest angle between embedding vectors, most often used
- euclid distance - (*Item B*) - similarity as inverse of distance of between embedding vectors terminal points
- dot product - (*Item A*) - straightforward but overestimates popular items over actual similar item

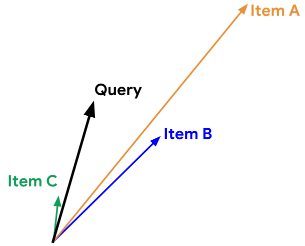


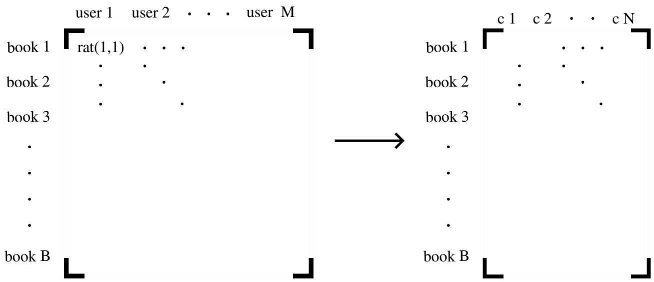
Figure: Embedding vectors (2D simplification)

Image source: https://developers.google.com/machine-learning/recommendation/overview/candidate-generation

Method 1: Principal Component Analysis

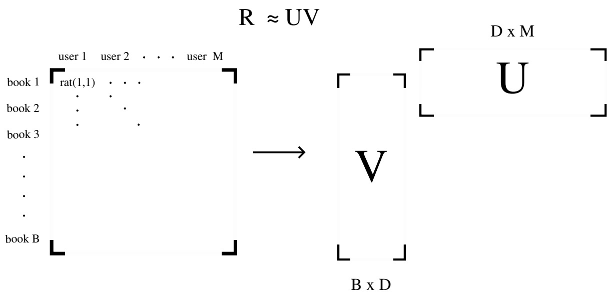
Creating book embedding vectors as compressed rows of item user matrix. Number of users M is substantially smaller than number of components N.

$$M >> N$$



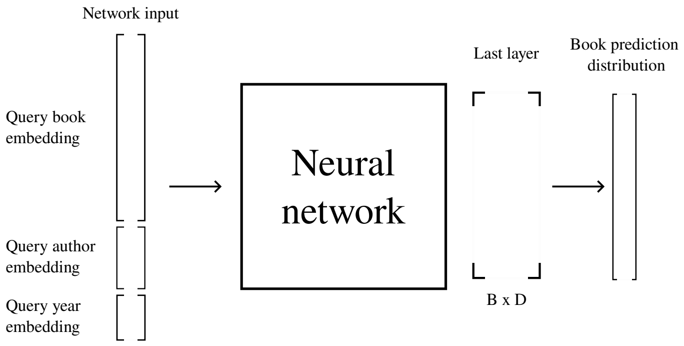
Method 2: Matrix factorization by gradient descent

Learning book embeddings U by minimizing objective MSE function for item user matrix R and its approximation UV. Starting embeddings are sampled from normal distribution.



Method 3: Matrix factorization by softmax model

Learning book embeddings as weights of last layer of SoftMax model. Neural network is learning to predict book as a cross entropy classification problem from input embeddings.



Retrieval evaluation

To evaluate quality of predicted recommendation it is necessary to have ground truth recommendations. For this reason for most rated books were generated pairs of (query book, List of recommended books), by nearest neighbor method used on original item user matrix.
Each method of book embedding creation had its quality of predicted recommendation evaluated against this evaluation data by measures of MAP, mean NDCG and MMR.

$$\mathrm{MAP} = \frac{1}{N} \sum_{q=1}^{N} \frac{1}{\mathrm{relevant}(q)} \sum_{k=1}^{K} P(k) \cdot rel(k)$$

$$\mathrm{mNDCG} = \frac{1}{N} \sum_{q=1}^{N} \frac{1}{\mathrm{DCG}_{\mathrm{ideal}}} \sum_{k=1}^{K} \frac{2^{rel(k)} - 1}{\log_2(k+1)}$$

$$\mathrm{MRR} = \frac{1}{N} \sum_{q=1}^{N} \frac{1}{\mathrm{rank}_{\mathrm{first}}}$$

Results

From implemented methods had best performance PCA embeddings for Nearest Neighbors using cosine similarity retrieval. All methods tried to learn book embeddings of same size B x D. Therefore PCA achieved embedding of more per book relevant information from item user matrix than other two methods.

Embedding method	**MAP**	**mNDCG**	**MMR**
PCA	0.421	0.52	0.795
Matrix factorization by gradient descent	0.089	0.151	0.334
Matrix factorization by softmax model	0.036	0.067	0.120

Possible (prototype) Improvements

- *Add new data*: Adding Goodreads, Amazon preview data to training data to improve quality of embeddings. (increased size of data, additional features, books from larger interval of years)
- *Query processing*: Get query book ID from partial query by finding closest book query embedding compared to partial query embedding that were gained by using tokenizers.
- *Better evaluation dataset*: Better testing of recommendation prediction generality by using custom made dataset from handpicked books that were higly rated by small group of users.
- *Modified similarity measure*: Multiplying cosine similarity with weighted dot product.
- *Modified gradient descent strategy*: Using Weighted Alternating Least Squares method to find user and book embeddings. (Converges in each step)
- *Different SoftMax model output*: Using multi label targets with against model output logits in Binary Cross Entropy could improve SoftMax model performance.

Ideal Recommendation System Scheme

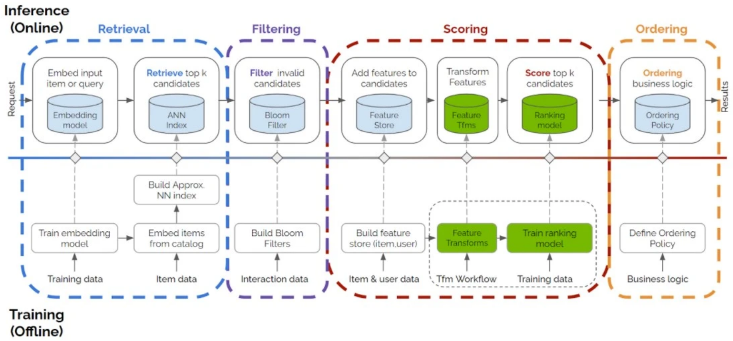
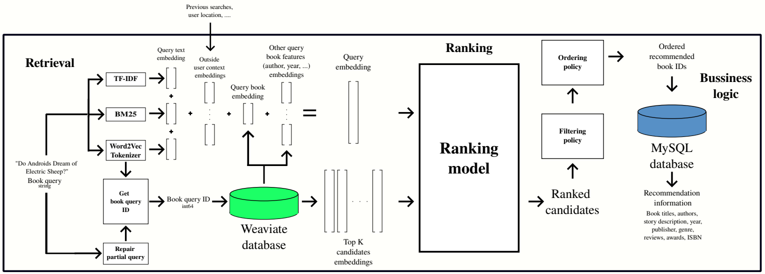


Image source: https://docs.nvidia.com/deeplearning/performance/recsys-best-practices/index.html

Recommendation System Scheme Proposal

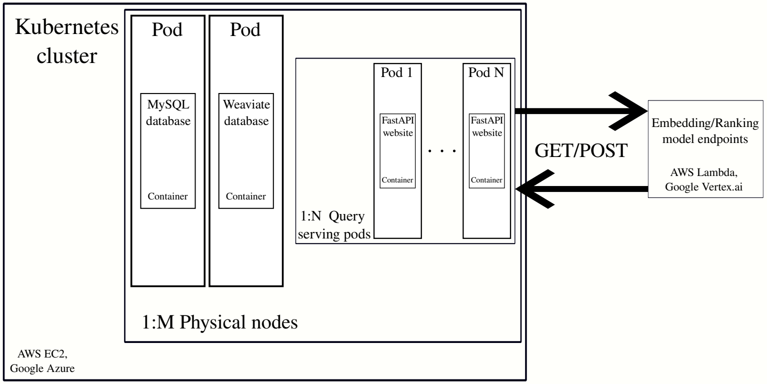


Previous searches, user location, ...
Retrieval
TF-IDF
BM25
Word2Vec Tokenizer
Query text embedding
Outside user context embeddings
Other query book features (author, year, ...) embeddings
Query book embedding
Query embedding
"Do Androids Dream of Electric Sheep?"
Book query string
Get book query ID
Repair partial query
Book query ID int64
Weaviate database
Top K candidates embeddings
Ranking
Ranking model
Ordering policy
Filtering policy
Ranked candidates
Ordered recommended book IDs
Bussiness logic
MySQL database
Recommendation information
Book titles, authors, story description, year, publisher, genre, reviews, awards, ISBN

Additional Proposal Information

- *TF-IDF, BM25* and *Word2Vec* - use for documents individual book titles.
- *Get book query ID* - Can be implemented by matching preprocessed query string against list of titles in MySQL database, or using Word2Vec tokenized raw query embedding to find nearest neighbor in tokenized titles within Weaviate database.
- *Weaviate* vector database Functions as a feature store, holding embedded vectors for various book aspects including the book itself, author, genre, reviews, and other features. It also provides precomputed nearest neighbors for immediate candidate embeddings retrieval, complete with relevant book features.
- *Outside context embedding* - Could be created for example by recurrent model.
- *New Data* - All embedding models (TF-IDF, BM25, Word2Vec, Book Embedding model) are periodically retrained after a batch of books is added. Candidate IDs for each book ID is precomputed at this step. The ranking model is also retrained but at less frequent intervals.

Product Deployment scheme



Product Deployment information

- Whether to use *Amazon* or *Google* services depends on knowledge base within team implementing solution. Both companies offer reliable services with world wide coverage and competitive pricing between themselves.
- *MySQL* was picked as faster to setup and less developer time to manage solution, when compared to *PostgreSQL*. NoSQL databases such as *MongoDB* are not needed as book information has rigid data structure.
- *FastAPI* is faster then *Flask*, *Django* and is recommended for use in ML applications.
- *Lambda* and *Vertex.AI* are compute oriented cloud services. So they offer minimal or zero cost when not processing query+candidate embeddings. This makes them good choice for model serving side of product.

Prototype demo

- Github: https://github.com/dobislukas/brs-app
- Prototype: https://brs-app.onrender.com/