F3

Faculty of Electrical Engineering
Department of Cybernetics

Bachelor's Thesis

Text Embeddings for Recommender Systems

Tomáš Černý

January 2024
Study program: Open Informatics
Specialization: Artificial Intelligence and Computer Science
Supervisor: Ing. Jan Drchal, Ph.D.

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Černý Tomáš** Personal ID number: **498948**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Text Embeddings for Recommender Systems

Bachelor’s thesis title in Czech:

Textové reprezentace pro doporučovací systémy

Guidelines:

The task is to experiment with various semantic text embeddings and their applicability to recommender systems:
1) Research state-of-the-art recommender systems and methods of text embedding extraction.
2) Choose an appropriate recommender system approach (such as EASE or ELSA) and several modern approaches to extract text embeddings (most likely Transformer-based contextual embeddings).
3) Find public datasets such as CiteULike, Epinions, or Netflix Prize (augmented by movie descriptions).
4) Research and design evaluation methodology.
5) Perform experiments measuring contribution of the embeddings to the quality of the recommendation.

Bibliography / sources:

[1] Ning, Xia, and George Karypis. "Slim: Sparse linear methods for top-n recommender systems." 2011 IEEE 11th international conference on data mining. IEEE, 2011.
[2] Vančura, Vojtěch, et al. "Scalable Linear Shallow Autoencoder for Collaborative Filtering." Proceedings of the 16th ACM Conference on Recommender Systems. 2022.
[3] Ghasemi, Negin, and Saeedeh Momtazi. "Neural text similarity of user reviews for improving collaborative filtering recommender systems." Electronic Commerce Research and Applications 45 (2021): 101019.
[4] Guo, Weiwei, et al. "Deep natural language processing for search and recommender systems." Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2019.

Name and workplace of bachelor’s thesis supervisor:

Ing. Jan Drchal, Ph.D. Artificial Intelligence Center FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **14.09.2023** Deadline for bachelor thesis submission: **09.01.2024**

Assignment valid until: **16.02.2025**

Ing. Jan Drchal, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. _____ Date of assignment receipt

_____ Student's signature

Acknowledgement / Declaration

I'd like to thank my family for support and friends for their company and showing me what *not* to do.

Also, I'd like to thank my supervisor for his guiding hand, advice and calm demeanor.

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague, on 9.1.2024, Tomáš Černý:

.....

Abstrakt / Abstract

V této práci zkoumáme způsoby, jak vylepšit existující doporučovací systémy pomocí textových embeddingů položek pro Top-N doporučování pomocí Sentence-BERT modelů.

Vyhodnocujeme zvolené metody vůči state-of-the-art baseline metodám na datasetu MovieLens 100k, který rozdělujeme na trénovací a testovací množinu podle času interakce. Pro ladění hyperparametrů využíváme vnořenou validaci.

Zjistili jsme, že jedna metoda s textovými embeddingy lehce zlepšuje kvalitu doporučení podle metrik na přítomnost položek (Hirate, Precision, Recall) a pořadí položek (NDCG) na zvoleném datasetu pro menší počet doporučovaných položek ale ne pro větší, kde je výkon srovnatelný.

Klíčová slova: doporučovací systémy, Top-N doporučování, textové embeddingy, Sentence-BERT, SBERT

Překlad titulu: Textové reprezentace pro doporučovací systémy

In this thesis, we explore ways to enhance existing recommender systems with items' Sentence-BERT textual embeddings for the Top-N recommendation task.

We evaluate the chosen methods against state-of-the-art baselines on the MovieLens 100k dataset split into test and train sets by interaction timestamp while using nested validation to tune hyperparameters.

We find that textual embeddings used in one of the chosen methods improve recommendation quality very slightly as measured by both item-presence (Hitrate, Precision, Recall) and item-ranking metrics (NDCG) on the chosen dataset for shorter recommendation lists but not for longer where performance is comparable.

Keywords: recommender systems, Top-N recommendation, text embeddings, Sentence-BERT, SBERT

Contents /

Tables / Figures

Chapter 1
Introduction

The amount of content and products available at users' fingertips is growing rapidly. Helping users find what they are looking for or might be interested in is crucial for their satisfaction and goals of the business. This is what recommender systems are for.

These recommender systems can sometimes be so good, users spend more time consuming content than they would like to. Examples of these platforms are YouTube, Instagram and TikTok which have effectively unlimited amount of content for a single user to see and rely heavily on their recommender system to increase time spent on the platform which is probably their main KPI (Key Performance Indicator).

If we take YouTube as an example, they are able to utilize a variety of information including user's geographic location, age, gender, past video watches, current search query and similar users' information to maximize user retention [1]. An example of their previosly used UI on a mobile app can be seen in 1.1.

Due to the value recommender systems can bring to a business, a lot of research is most likely kept behind closed doors. Meanwhile, academic research can suffer from low reproducibility and reliability of results as demonstrated by [2]. One of the factors is the plethora of choices when designing the evaluation procedure of a recommender system for a particular dataset. This is summarized in figure 1.2. We'll therefore attempt to devise a fair evaluation strategy since we won't have a real service or platform to experiment on.

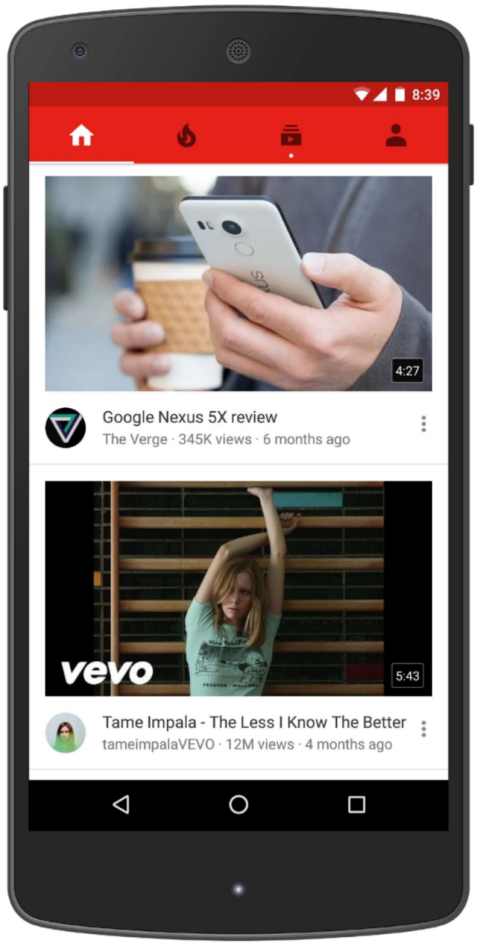

Figure 1.1. Recommendations displayed on YouTube mobile app home. Image and caption from [1].

Recent advances in NLP (Natural Language Processing) based on the transformer architecture from [4] make it possible for computers to understand human language [5] and reproduce it [6] far better than before. One of the downstream advances is encoding not just words but text into high-quality fixed-sized vectors called embeddings which have numerous applications like STS (semantic textual similarity) [7].

As we will see, using only textual embeddings to recommend yields rather poor results, at least on the data we've tried. Utilizing collaborative information alone

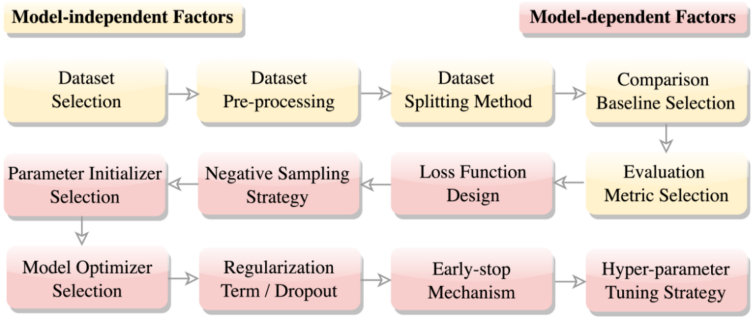

Figure 1.2. Hyper-factors within the whole recommendation evaluation chain. Image and caption from [3].

without any textual processing has much better performance. In this work, we attempt to improve collaborative recommendation using textual embeddings of the recommended items. The methods we have as baselines are Top-Pop, Collaborative ItemKNN, UserKNN, and EASE^R. As extensions with textual embeddings we have Content-based and Hybrid ItemKNN, and Hybrid EASE^R.

There are two main motivations for this work. First, understanding the basics of recommendation might come in handy in the author's entrepreneurial endeavors regardless of whether a SaaS (Software as a Service) is used or a custom implementation is chosen. This understanding might provide a notion of the how and why of a recommendation service and how real world platforms' recommendation operates, at least on a superficial level. Second, although the author's study program includes basics of machine learning and artifical intelligence, unfortunately, no NLP techniques were taught in any of the mandatory subjects. Understanding and using textual embeddings and the models producing those embeddings on a concrete task is a first step worth taking.

This work is structured as follows — first we introduce basic recommendation terms and what kind of recommendation we're going to perform in chapter 2, then we delve into details regarding of state-of-the-art textual embedding models in chapter 3, next we describe ways of "marrying" recommendation and textual embeddings in chapter 4. Then we describe our evaluation methodology and its rationale in chapter 5 and in chapter 6, we report on our experimental setup and its results where we compare baseline recommendations with appraoches from chapter 4. Lastly in chapter 7, we conclude our results and discuss what might be worth exploring in the future.

Chapter 2
Recommendation

2.1 Recommendation Tasks

In this work, we're going to focus solely on **Top-N recommendation**. In Top-N recommendation, given a set of users U which interact with items I, the task is to recommend a list of N items from I to each user $u \in U$.

There are other recommendation tasks such as content-aware recommendation where we might utilize the current search query, user's location or time of day, and cold-start recommendation where the goal is to recommend items to new users or items with no historical interactions.

We're also going to focus on one type of item. For example, in e-commerce setting, many types of items are offered for purchase such as books, electronics, apparel, etc. We'll focus on one type of item such as books.

2.2 Content-Based & Collaborative Filtering

In **content-based filtering**, we utilize user and item attributes to make recommendations [8, p. 133].

For example, a movie streaming service might ask a user their favourite genres and recommend movies in these genres preferring movies that match all of the user's favourite genres.

In **collaborative filtering**, we utilize interactions between users and items and thus don't require user or item specific information [8, p. 91].

This means we can avoid feature engineering and apply CF algorithms generally to many domains without modification.

Approaches that combine CF and CBF are known as **Hybrids** [8, p. 133].

2.3 User-Item Matrix

We have a set of users U that interact with items I according to values in matrix $\mathbf{X} \in \mathbb{R}^{|U| \times |I|}$. To shorten notation, we'll be using $m = |U|$ and $n = |I|$. Each row represents feedback of one user and each column represents feedback for one item.

2.4 Explicit & Implicit Feedback

In recommendation, we work with two kinds of feedbacks, explicit and implicit [8, p. 143].

An example of explicit feedback is user's star rating on a movie or product. In those cases, we might call the user-item matrix $\mathbf{X}$ "rating matrix".

Implicit feedback is a mere interaction with an item such as watching a movie or purchasing a product. In those cases, we might call the user-item matrix $\mathbf{X}$ "interaction matrix".

Implicit feedback is more commonly used in recommendation models even when the original data has explicit feedback. This is done by mapping explicit feedback such as star rating via a function $f(r) = \begin{cases} 1 & \text{if } r \geq 4, \\ 0 & \text{otherwise} \end{cases}$. Note that 0 has both meanings of "user hasn't interacted with the item" and "user doesn't like the item".

2.5 Types of Methods

Two broad groups of methods are commonly used: (1) neighbourhood-based methods and (2) model-based methods [9].

The first group includes methods such as ItemKNN [10] (see 2.6.2). ItemKNN is sometimes referred to as a memory-based method along with popularity based Top-Pop (see 2.6.1) like in [11].

The second group is where all the fuzz is. However, it's further division into subgroups isn't clear cut and methods can have elements of more than one subgroup. In [12] model-based methods includes latent factor models, autoencoders and deep-learning methods.

Latent factor models learn a "hidden" representation of items and users such as Matrix Factorization (MF) [13–15] and Factorization Machines (FM) [16].

Autoencoders include MultVAE [17], SLIM [9], EASE^R [18] (see 2.6.4) and ELSA [12]. However, the author of SLIM draws similarities between it and matrix factorization models as SLIM learns an item-item weight matrix and uses the original interaction matrix as user representations.

As for deep-learning methods, there is NeuMF [19], Neural Factorization Machines (NFM) [20] and many others. The two mentioned share similarities with latent factor models in that they learn embeddings.

2.6 Used Methods

2.6.1 Top-Pop

For each item, we have a counter. By summing the number of non-zero columns of the user-item matrix from the training set, we obtain each item's popularity. Every user then gets the same recommended list (non-personalized) which we obtain by sorting item popularities in descending order. Last we truncate to get the top N [2].

2.6.2 ItemKNN

This simple algorithm is inspired by the classic K-Nearest Neighborhs (KNN) algorithm. We need to define a similarity function *sim(i, j)* outputing a real number measuring the similarity of items i and j. [10] An example of a similarity function is the cosine angle of column vectors from the user-item matrix $\mathbf{X}$.

In short, for every item j, it finds K most similar items according to *sim(i, j)* except for $sim(i, i)$ which it ignores by setting it to zero. When recommending, it sums scores of all items a user has interacted with and recommends top N items the user hasn't interacted with yet.

We'll adapt the definitions from [10] to match our notation and pseudocode to match the rest of this text. We also add comments regarding the purpose of certain operations for clarity's sake.

Algorithm 2.1. Build ItemKNN Model(interaction matrix $\mathbf{X}$, number of neighbours k)

```
M = zeros(n, n)

for j in items:
    for i in items:
        if i == j:
            continue
        M[i, j] = sim(i, j)

    for i in items:
        if not is_item_among_k_largest(M[:, j], M[i, j], k):
            # zero-out items not in `k` nearest neighbours
            M[i, j] = 0

return M
```

Output is a learned square matrix $\mathbf{M}$ with one column for each item.

Algorithm 2.2. Apply ItemKNN Model($\mathbf{M}$, vector of user's interacted items $\mathbf{u}$, N)

```
x = M @ U

for j in items:
    if U[j] != 0:
        # zero-out items the user has already interacted with
        x[j] = 0

for j in items:
    if not is_among_N_largest(x, x[j], N):
        # zero-out items not in `N` most relevant items
        x[j] = 0

return x
```

Output is a vector $\mathbf{x}$ with a score for each item with exactly N scores being non-zero.

2.6.3 UserKNN

Using other users for prediction was probably first seen in [21] which used Pearson correlation of two users' ratings to compute how they "agree" with each other. Now it's common to use cosine similarity of users to measure how much their tastes agree like in [22] and implementation of [2].

Computation-wise, it's very similar to ItemKNN 2.6.2. K most similar users are found based on their interactions or ratings via a similarity function. To predict, find the K most similar users and multiply their respective ratings or interactions by how similar the user is, exclude already seen items and return the top N items.

2.6.4 EASE^R

EASE^R stands for Embarassingly Shallow Auto-Encoder (when reversed) and it's meant for sparse data [18]. It learns an $n \times n$ square matrix $\mathbf{B}$ by optimizing the following convex objective function with λ as a regularization parameter that has to be tuned.

$$\begin{aligned} \min_{B} \quad & \|\mathbf{X} - \mathbf{X}\mathbf{B}\|_F^2 + \lambda \|\mathbf{B}\|_F^2 \\ \text{s.t.} \quad & \operatorname{diag}(\mathbf{B}) = 0 \end{aligned} \tag{1}$$

The condition on diagonals of $\mathbf{B}$ ensures that an item isn't used to predict itself and so the model is forced to rely on other items. The author of EASE^R used Lagrangian multipliers to arrive at the following closed-form solution:

$$\mathbf{P} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I})^{-1} \tag{2}$$

$$\mathbf{B}_{i,j} = \begin{cases} 0 & \text{if} \quad i = j, \\ \frac{-\mathbf{P}_{ij}}{-\mathbf{P}_{jj}} & \text{otherwise} \end{cases} \tag{3}$$

EASE^R doesn't use a non-negativity constraint on the learned weights so it can capture dissimilarity. Otherwise it would be the same as SLIM which uses gradient optimization as there is no closed-form solution [9].

One downside of EASE^R is it's memory requirements which grow with $\mathcal{O}(n^2)$ due to dimensions of $\mathbf{B}$ and $\mathbf{X}^T\mathbf{X}$. In [12] they've drastically lowered the memory requirements by learning a matrix $\mathbf{A}$ of size $n \times r$ where r is tunable. Matrix $\mathbf{A}\mathbf{A}^T$ is then the equivalent of $\mathbf{B}$ from EASE^R.

However, we'll be using EASE^R as it doesn't require gradient optimization and ELSA's performance gains over EASE^R aren't substantial.

Chapter 3
Textual Embeddings

A common and fairly universal technique for many tasks is to encode data that's hard for a computer to understand directly such as text and images to fixed-size vectors. Once precomputed, they can be used without high computing power to power various tasks such as semantic search and classification.

3.1 Word-level

In Word2Vec [23], the authors proposed two new models, Continuous Bag-of-Words Model (CBOW) and Continuous Skip-gram Model for learning vector word representations. The former learns using a context window of words to predict the middle word and the latter learns using a middle word to predict it's surrounding context. Both can be considered a single-layer neural networks. Due to their simplicity over feed-forward and recurrent neural networks, they could train the models on a much larger data set (at the time) while still achieving high quality vectors.

GloVe [24] used co-occurrence probabilities of words which [23] use did not. It trains a matrix factorization model with bias terms which had outperformed other proposed models while being able to learn fast.

3.2 Sentence-level

SIF [25] and Siamese-CBOW [26] produce sentence embeddings via weighted averages of word embeddings and [27] averages learned n-gram embeddings to yield sentence embeddings.

Language representation model BERT [5] trained on a bidirectional context achieved state-of-the-art performance on various NLP tasks with the transformer architecture from [4] with its attention mechanism. BERT is trained on the masked language modeling (MLM) task where input is corrupted with 15% probability and the model is tasked to predict the original text. Another training task used in BERT is the next sentence prediction (NSP) task where the model is trained to predict whether two input sentences are consecutive in some larger text.

It has been shown that pre-training a language model on a large dataset and further finetuning and augmenting the model to a specific task yields better results than training a model from scratch [28–31].

Using BERT with finetuning only is unsuitable for sentence-pair regression tasks such as sentence similarity at scale due to high computational costs. A sentence-pair has to be passed to BERT with a `[SEP]` token between them. This has to be done for every sentence-pair which scales with $\mathcal{O}(n^2)$ [7].

Sentence-BERT (SBERT) [7] uses a siamese architecture finetuned on NLI data [32] to train for sentence similarity. At inference time, output vector of the last pooling layer is used as the sentence embedding. See figures 3.1 and 3.2. SBERT achieves

state-of-the-art performance on semantic textual similarity (STS) tasks outperforming InferSent [33] and Universal Sentence Encoder [34].

BERT has downstream models such as DistilBERT [35] and RoBERTa [36]. There are also models which borrowed some ideas from BERT such as MPNet [37]. Textual embeddings can also be finetuned on a specific task such as semantic search or question answering. Another dimension is whether the model is multilingual (explored in [38]) or not, its suitable similarity, size and therefore speed. These models are publicly available via a Python package[1]. A few example models are `all-mpnet-base-v2`, `multi-qa-distilbert-cos-v1`, and `paraphrase-multilingual-mpnet-base-v2`.

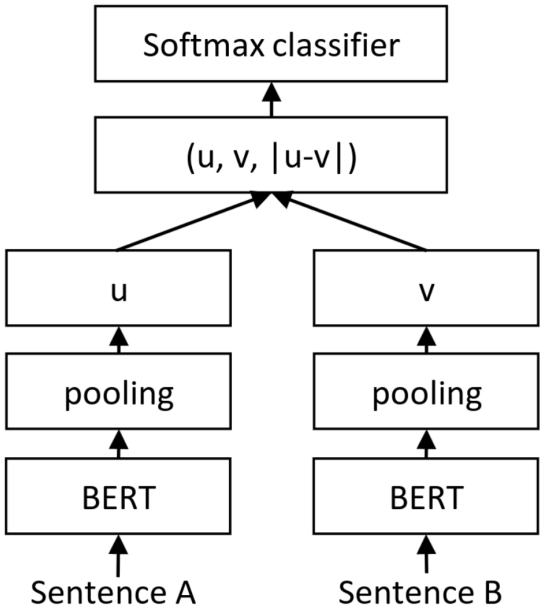

Figure 3.1. BERT architecture with classification objective function, e.g., for fine-tuning on SNLI dataset. The two BERT networks have tied weights (siamese network structure). Image and caption from [7].

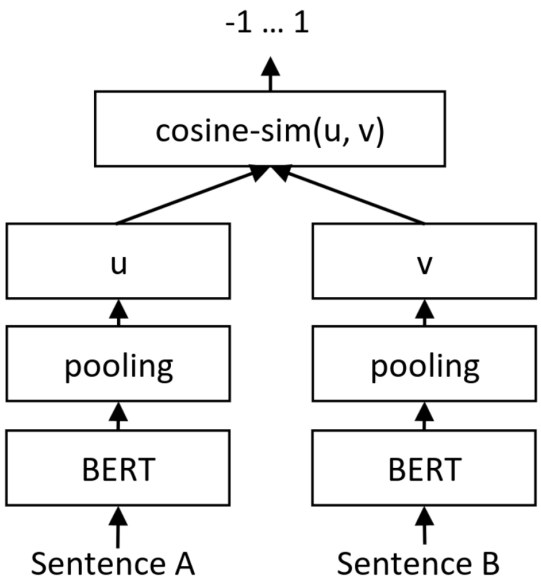

Figure 3.2. BERT architecture at inference, for example, to compute similarity scores. This architecture is also used with the regression objective function Image and caption from [7].

3.3 Beyond Sentences

SBERT models have input size limitation. The main factor is that transformers from [4] have a quadratic time complexity with regards to input length. According to SBERT's documentation[2], the default input size limit is 128 tokens although some of the models can accept longer inputs when configured to do so. For longer inputs, only the prefix tokens are considered. They mention that 512 tokens is around 300-400 words in English, so 128 tokens should be around 75-100 words. Since there is vast choice of models and recommenders have hyperparameters to tune, we don't play with the limit in this work to not increase the search space further and rather focus on model selection.

[1] `https://sbert.net`

[2] `https://sbert.net/examples/applications/computing-embeddings/readme.html#input-sequence-length`

Chapter 4
Marrying Recommendation with Textual Embeddings

4.1 Approaches Without Embeddings

Attempts at using textual information in recommendation were made even before Word2Vec [23]. In [39], the authors used multiple bags-of-words (title, cast, etc.) with a naive Bayesian classifier to enhance collaborative filtering and called it content-boosted collaborative filtering. A user vector would contain all the user's ratings. A score from the content-based predictor would serve as fallback making user vectors dense. Then, a UserKNN neighbourhood-based approach with Pearson correlation as similarity between users was applied.

More recently, using user reviews to extract topics was explored in [40–41]. [42] used attention-based CNNs. The input to their network is all of user's reviews as the user document and all of item's reviews as the item document. In their approach, certain words in reviews were informative as their learned model to pay higher attention to them. [43] also used review texts and convolution but not any kind of attention and performed worse than [42] on both Yelp and Amazon Music Instruments datasets in terms of root mean square error (abbreviated as RMSE, see 5.3.3).

However, training these big networks is computationaly expensive and not all datasets contain user reviews. [42] used multiple GPUs to train and optimize their neural network.

In [44], they measured user similarity by maximum similarity of reviews of a pair of users in the first module and in the second module, they used cosine similarity of users' ratings. Output of these two modules was combined with equal weights to produce rating prediction. In the review similarity module, they used seven methods of which the best one was cosine similarity of LSTM-based textual embeddings. The other six methods included TF-IDF and Word2Vec.

4.2 Why Embeddings?

One of the reasons recommendation techniques often use implicit feedback is that explicit feedback in the form of ratings and reviews is harder to come by in great quantities.

In many recommendation scenarios, we have textual (and visual) information about the items, be it a movie's title or a product's description. As outlined in 3.2, there are easily available high quality sentence-level embeddings. The only missing piece then is the *how* of using them in recommendation.

4.3 Using Textual Embeddings in Recommenders

4.3.1 Content-based ItemKNN

The simplest way to recommend with textual embeddings is finding items similar to those a user has already interacted with or rated positively using the embeddings. We will refer to this as Content-based ItemKNN [45]. Instead of measuring item-item similarities by their user interactions like in 2.6.2, we can use textual embeddings of item title or description in the similarity function. The exact similarity function depends on the used embedding model, some are suitable with dot product, some with euclidean distance and some with cosine angle.

One obvious issue is that information about other users isn't used at all as this is a purely content-based method. Users might have problems discovering relevant items that are "dissimilar enough" from their historical interests.

4.3.2 Hybrid ItemKNN

[2] described ItemKNN-CFCBF (Collaborative Filtering & Content-based Filtering) where for each item i we concatenate a vector of ratings/interactions and item features with a weight — $[\mathbf{r}_i, w \cdot \mathbf{f}_i]$. To avoid long abbreviations, we'll refer to this method as Hybrid ItemKNN. We'll also use a weight $w = 1$.

4.3.3 EASE^R with textual embeddings

How can we add item features to EASE^R? Given item-specific features $\mathbf{F}_I \in \mathbb{R}^{n \times f_i}$, we can create an augmented matrix $\mathbf{X}'$ like in Hybrid ItemKnn (4.3.2):

$$\mathbf{X}' = \begin{bmatrix} \mathbf{X} \\ \mathbf{F}_I^T \end{bmatrix} \in \mathbb{R}^{(m+f_i) \times n}$$

If we expand EASE^R's closed-form solution in equation (2) for $\mathbf{P}$, we get

$$\mathbf{X'}^T\mathbf{X}' = \mathbf{X}^T\mathbf{X} + \mathbf{F_I}^T{}^T\mathbf{F_I}^T = \left(\mathbf{X}^T\mathbf{X} + \mathbf{F_I}\mathbf{F_I}^T\right) \in \mathbb{R}^{n \times n}$$

We're essentially adding similarities of items onto item co-occurrence matrix $\mathbf{X}^T\mathbf{X}$.

4.3.4 Failed Experiments

We've experimented with two-tower neural network architecture similar to [1] where one of the inputs was the average of items' textual embeddings that a user has interacted with. However, we completely failed to make this work — our metrics (which we'll see in 5.3) were 0 even during training, binary cross-entropy jumped quite a lot even for very low learning rates and multiple optimizers. Due to that we've decided to abandond this approach.

Chapter 5
Evaluation Methodology

5.1 Reliability of Results

In recent years, neural recommendation approaches came under critique due to several reasons. (1) They've compared to weak baselines, (2) they've compared to weak methods that had compared to weak baselines themselves, and (3) their results are difficult to reproduce [2].

The neural approaches benchmarked in [2] includes Collaborative Varational Autoencoder (CVAE) [46] presented at KDD '18, Neural Collaborative Filtering (NCF or sometimes NeuMF) [19] presented at WWW '17 and Spectral Collaborative Filtering (SpectralCF) [47] presented at RecSys '18.

5.1.1 Simplicity over Complexity

Authors of [2] demonstrated better performance than several neural approaches by tuning relatively simple algorithms. This included ItemKNN (2.6.2), UserKNN (2.6.3), SLIM (similar to EASE^R, 2.6.4) and trivial Top-Pop (2.6.1) which worked best for one dataset better than relatively complicated neural approaches.

A conclusion that can be drawn from this is that simple algorithms can work very well when tuned.

5.2 Offline vs Online

Online evaluation works by A/B testing where each user can be routed to a different recommender and the better one is chosen using some metric such as Click Through Rate (CTR). When evaluating offline, we measure how well a recommender does on a specific dataset [48].

In online evaluation there are, of course, other metrics than CTR such as video watch time and article reading time. These can be considered business KPIs (key performance indicators) that a stakeholder might be intersted in.

There are many metrics for offline evaluation too. We'll see some of them in 5.3. In this work, we'll be focusing solely on offline evaluation as it doesn't require a real site or service.

5.3 Defining Offline Metrics

Given a list of recommended items and a list of relevant items for a user, we can assign items into 4 sets as seen in table 5.1.

Below we'll be using abbreviations *tp* for true positive, *fp* for false positive, *fn* for false negative and *tn* for false negative.

	relevant	¬relevant
recommended	true positive	false positive
¬recommended	false negative	true negative

Table 5.1. Sets of recommended and relevant items.

5.3.1 Item Presence Metrics

The definitions below are useable for a single user. To compute them for multiple users, we would average them.

Hitrate is 1 if there is at least one relevant recommended item and 0 otherwise. Hitrate $= \begin{cases} 1 & \text{if } |\text{tp}| > 1, \\ 0 & \text{otherwise} \end{cases}$.

Precision is the ratio of recommended items that are relevant. It's value is computed by Precision $= \frac{|\text{tp}|}{|\text{tp}|+|\text{fp}|}$.

Recall is the ratio of relevant items that are recommended. It's value is computed by Recall $= \frac{|\text{tp}|}{|\text{tp}|+|\text{fn}|}$.

5.3.2 Ranking Metrics

Now we'll build a definition for Normalized Discounted Cumulative Gain (NDCG) [49] piece by piece.

Cumulative Gain (abbreviated as **CG**) is computed by a prefix sum $\sum_{i=1}^{N} G_i$. It has the obvious disadvantage that every item, even those at the far end, contribute the same as the first few items. It's sometimes referenced to as Direct Cumulative Gain.

Discounted Cumulative Gain (abbreviated as **DCG**) is computed by

$$\sum_{i=1}^{N} \frac{G_i}{\log_2 i + 1}$$

It improves over CG by discounting items that are farther down the list even though they might have high relevance. However, this still doesn't allow us to compare two lists since we have no information about which items *aren't* in the result list.

Idealized Discounted Cumulative Gain (abbreviated as **iDCG**) is the max possible DCG. We can obtain it by sorting relevances of all items and keeping the first N.

Normalized Discounted Cumulative Gain (abbreviated as **NDCG**) is a ratio of DCG of a result list and iDCG by computing $\frac{\text{DCG}_N}{\text{iDCG}_N}$.

Although NDCG is able to work with different revelancy values, we'll be using binary values. Relevant items will have relevance equal to 1 and irrelavant items will have relevance equal to 0. Irrelavant items includes those a user has already interacted with (interactions are in the training set) and items a user has not interacted with in the whole dataset.

5.3.3 Rating Error Metrics

In recommendation tasks predicting users' ratings, we might want to know what's the average error, e.g. the average number of stars a recommender is off by from the real rating.

Root Mean Square Error (abbreviated as **RMSE**) is computed by

$$\sqrt{\frac{\sum_{i=1}^{T} (\hat{r}_i - r_i)^2}{T}}$$

where T is the number of ratings, r_i is the i-th ground truth rating and $\hat{r}_i$ is the i-th predicted rating.

5.4 Benchmarking Recommendation by DaisyRec

According to [3], evaluation methodologies differ wildy in recommender system research. A minority of 4% out of 141 the papers they've studied don't provide important details such as how they split datasets into train and test sets. 21% of papers don't specify data filtering methods (e.g. ignoring users with less than 5 interactions) and more than 50% don't report parameter initialization methods (e.g. Normal distribution or Xavier initialization).

In the fourth chapter "Benchmarking recommendation" of [3], the authors conclude the following recommendations on fair, reproducible and reliable evalution:

1. Evaluate on at least one public dataset for reproducibility.
2. Report used **filtering** of users and items with few interactions as it affects performance but helps with sparsity.
3. Prefer time-aware **data splitting** into train & test sets as random splitting approximates real recommendation scenarios better.
4. For **hyperparmeter tuning**, they claim a nested validation is mandatory. Bayesian HyperOpt for intelligent search is recommended.
5. Choose **representative baselines** - they hint at using at least one memory-based method, latent-factor model and deep-learning model.
6. Adopt at least one **metric** measuring whether an item is present in the top-N recommendation list (e.g. Precision) and one measuring ranking positions of items (e.g. NDCG).
7. Use the same **objective function** with different methods (e.g. cross-entropy loss, BPR [14]) to measure each method's contributions.
8. Consistently use the same **parameter initializer** and **optimizer**.
9. Use the same **negative sampling** approach. This can refer to negative sampling used while training (e.g. using uniform sampling, popularity-based sampling) and negative sampling used in evaluation. The claim in [3] is that 1000 negative samples during evaluation is enough while speeding up the evaluation.
10. Use the same **overfitting prevention** strategies (e.g. early-stop).
11. **Source code** should be available. Conferences could make them required.

We'll therefore specify our evaluation methodology and all of its aspects with great detail. Note that some algorithms don't require parameter initialization (e.g. EASE with it's closed form solution) and some don't have a hyperparmeters (e.g. Top-Pop).

Chapter 6
Experiments

6.1 Setup

Here's the overview of our choices regarding evaluation and our reasoning. It follows the same numbering as in 5.4.

1. We evaluate on ml-100k [50], which is a public dataset[1].
2. We **filter** out users with less than 5 interactions and items with no interactions. Datasets are sometimes already prefiltered and our filtering is therefore a no op as is the case for ml-100k.
3. We use **Time-aware Split-By-Ratio** (TSBR) as recommended by [3] as the data splitting method. This should approximate real-world scenarios better where we try to predict the next item a user will interact with using past information. The other recommended choice was Time-aware Leave-One-Out (TLOO). Both imply weak generalization [18] where splits share users.

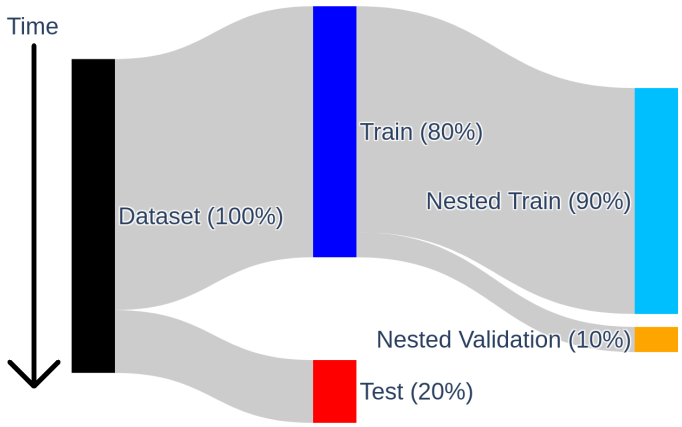

Figure 6.1. Time-aware Split-By-Ratio (TSBR) of user interactions

4. We use **nested validation** for hyperparmeter tuning using the same TSBR approach with 90% of interactions in the nested train set (out of the 80% of the whole) and 10% in the nested validation set. Although the reasoning for nested validation in [3] isn't justified, it makes sense to do as we have more training data than if we would have distinct train, validation and test splits.
5. We choose Top-Pop (see 2.6.1) and every algorithm we try to extend with textual embeddings as **baselines**. We're therefore missing a true latent-factor method and any deep-learning method.

[1] `https://grouplens.org/datasets/movielens/100k`

6. We use **precision** and **recall** together to study the tradeoff between them for different lengths of the recommendation list as well as **hitrate** for item presence. For measuring ranking, we use **NDCG**. This follows the recommendation in [3] to use both item presence and ranking metrics.
7. We don't choose any **objective function** since the used algorithms don't require any.
8. We don't choose any **parameter initializer** nor **optimizer** since the used algorithms don't require any.
9. We don't use any **negative sampling** and both train and evaluate on the whole dataset or split.
10. We don't use any **overfitting prevention** strategies beside what the algorithms already provide e.g. via regularization.
11. Making the **source code** available is a requirement for this work.

6.2 Dataset Details

See table 6.1 for a summary of the used dataset and table 6.2 for text field descriptions. It's important to note that no movie has title longer than what any textual embedding model is capable of handling. For movie plot this doesn't hold true but only $\leq 1\%$ of movies have plot longer than the limit of around 75-100 English words.

The process of obtaining movie plots from IMDB was quite elaborate as links in the dataset aren't functional. Some movies have a slightly different title in the dataset and on IMDB so the process was partly automatic and partly manual. For a minority of movies, no plot was found and movie title with release year is used instead as fallback.

Sparsity of a dataset s is calculated as

$$s = \frac{1}{|U||I|} \sum_{(u,i) \in U \times I} \gamma_{u,i}$$

$$\gamma_{u,i} = \begin{cases} 1 & \text{if } \mathbf{X}_{u,i} = 0, \\ 0 & \text{otherwise} \end{cases}$$

dataset	time	# users	# items	# interactions	sparsity
ml-100k	yes	943	1 682	100 000	93.70%
filtered ml-100k	yes	942	1 447	55 375	95.94%

Table 6.1. Dataset summary. Filtering refers to ignoring users with less than 5 interactions and items with no interactions after the first filtering step.

field	source	q{50,99,100}(# words)	example
title (with year)	dataset	4, 10, 16	Toy Story (1995)
plot	IMDB, fallbacks to title	25, 69, 181	A cowboy doll is profoundly threatened...

Table 6.2. ml-100K text field summary. The third column refers in order to 50^{th} percentile, 99^{th} percentile and 100^{th} percentile (max) of the number of words.

6.3 Implementations

For both UserKNN and ItemKNN, we used our implementations based on the respective sources although the original sources would use a different similarity metric. For EASE^R, we used the implementation from [3] available at [11][2] which we modified to suit our input and needs.

6.4 Tuning Baselines Without Textual Embeddings

All baselines were tuned using the same nested splits.

For EASE^R, we tried

$$\lambda \in \{0.01, 0.1, 1, 10, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 600, 700, 800, 900, 1000, 1200, 1400, 1600, 1800, 2000, 2500, 3000, 3500, 4000, 4500, 5000\}$$

and chose $\lambda = 250$ which performed best. How NDCG@10 changed with λ is displayed in figure 6.2.

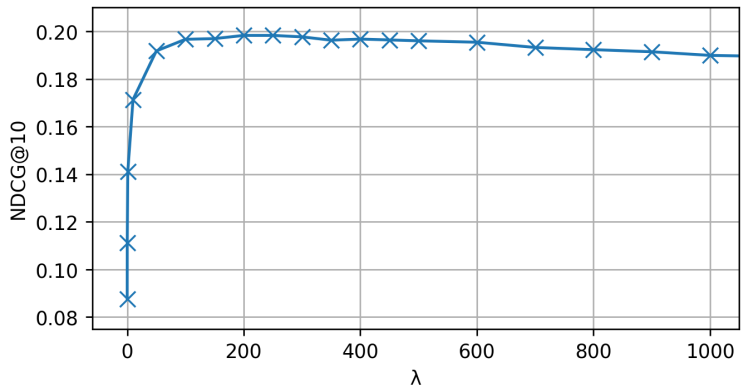

Figure 6.2. Nested evaluation of EASE on ml-100k for different values of λ. Values $\lambda > 1000$ are not displayed since the values are consistently lower and lower.

For collaborative ItemKNN and UserKNN, we tried $K \in \{1, 2, \ldots, 100\}$ with dot and cosine similarity. We chose $K = 14$ with cosine similarity for ItemKNN and $K = 20$ with cosine similarity for UserKNN. How NDCG@10 changed with K and similarity is displayed in figure 6.3.

6.5 Tuning Methods With Textual Embeddings

We'll choose an embedding model from the SBERT family. The criterion will be again NDCG@10 on nested train & validation splits.

[2] `https://github.com/recsys-benchmark/DaisyRec-v2.0/blob/dev/daisy/model/EASERecommender.py`

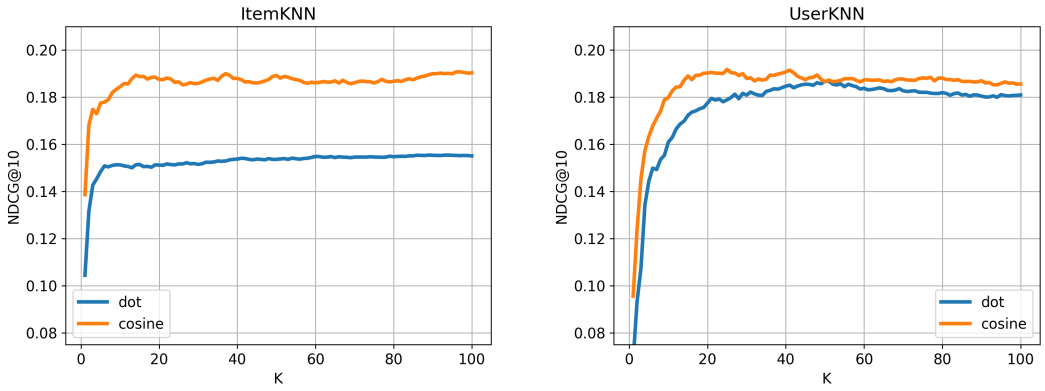

Figure 6.3. Nested evaluation of ItemKNN (left) and UserKNN (right) on ml-100k for different values of K and similarity.

field	K	best model	similarity	NDCG@10
title	5	`all-mpnet-base-v2`	cosine	0.0780
	10	`all-mpnet-base-v2`	cosine	0.0735
	15	`multi-qa-mpnet-base-dot-v1`	cosine	0.0745
	20	`multi-qa-mpnet-base-dot-v1`	cosine	0.0758
	25	`all-mpnet-base-v2`	cosine	0.0711
	30	`all-mpnet-base-v2`	cosine	0.0693
plot	5	`distiluse-base-multilingual-cased-v2`	cosine	0.0340
	10	`smarco-bert-base-dot-v5`	dot	0.0323
	15	`msmarco-distilbert-cos-v5`	dot	0.0301
	20	`msmarco-bert-base-dot-v5`	cosine	0.0298
	25	`msmarco-bert-base-dot-v5`	cosine	0.0293
	30	`msmarco-bert-base-dot-v5`	cosine	0.0282

Table 6.3. Nested evaluation of embedding models in Content-Based ItemKNN for each field and $K \in \{5, 10, 15, 20, 25, 30\}$ on ml-100k dataset. Only best embedding model is shown.

6.5.1 Choosing an Embedding Model for ml-100k

Initially, we tried to select an embedding model, field and similarity function for all methods using performance of Content-based ItemKNN with that embedding model.

In table 6.3, we can observe that (1) using the title field is generally better than using the plot field, (2) with increasing K, the performance doesn't increase, (3) cosine similarity is overall better than dot product, (4) there is also an unexpected result that models finetuned with dot product sometimes perform better with cosine similarity (field plot, $K \in \{20, 25, 30\}$).

For evaluating Content-based ItemKNN, we'll use `multi-qa-mpnet-base-dot-v1` with title field in the final evaluation since it had the highest NDCG@10 for $K = 20$.

Based on the previous data, we came to the conclusion of using both `all-mpnet-base-v2` and `multi-qa-mpnet-base-dot-v1` for the title field with cosine similarity would be best for other methods.

However, when running Hybrid ItemKNN with those embedding models, we found that `multi-qa-mpnet-base-dot-v1` performed way worse in nested evaluation than the other model `all-mpnet-base-v2` which can be seen in figure 6.4. This indicates

that using Content-based ItemKNN (where they performed simiarly although for different K) is not a suitable proxy for choosing which embedding model to use in Hybrid ItemKNN and possibly other methods. Maybe even the choice of the title field is invalid.

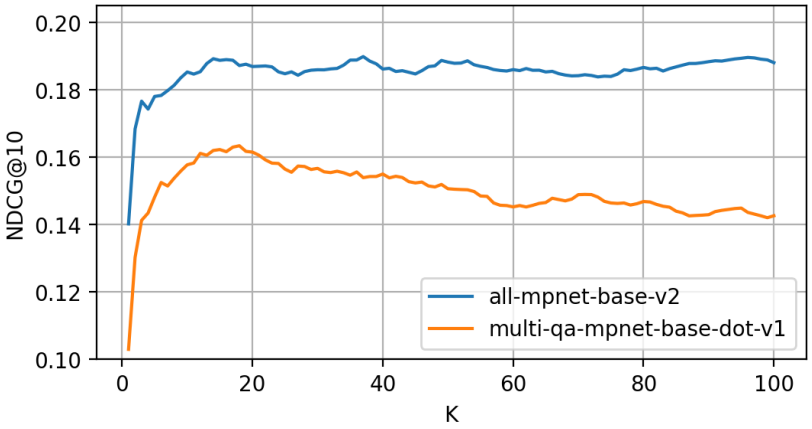

Figure 6.4. Nested evaluation of Hybrid ItemKNN on ml-100k for two embedding models using title and cosine similarity.

We therefore ran another set of experiments for Hybrid ItemKNN for all combinations of 2 fields, 21 models, and 2 similarities totalling 84 experiments on the nested train & validation splits. To save computing resources, we only tried $K \in \{1, 2, \ldots, 50\}$ as it seemed unlikely that higher values of K would perform better. The best performing model was in the group using plot and cosine similarity whose results can be seen in figure 6.5. We can see that most embedding models performed well with few exceptions which includes the previously chosen `all-mpnet-base-v2` and `multi-qa-mpnet-base-dot-v1`. The best result was achieved by `msmarco-distilbert-cos-v5` using the plot field at $K = 16$ which is what we'll use for the final evaluation.

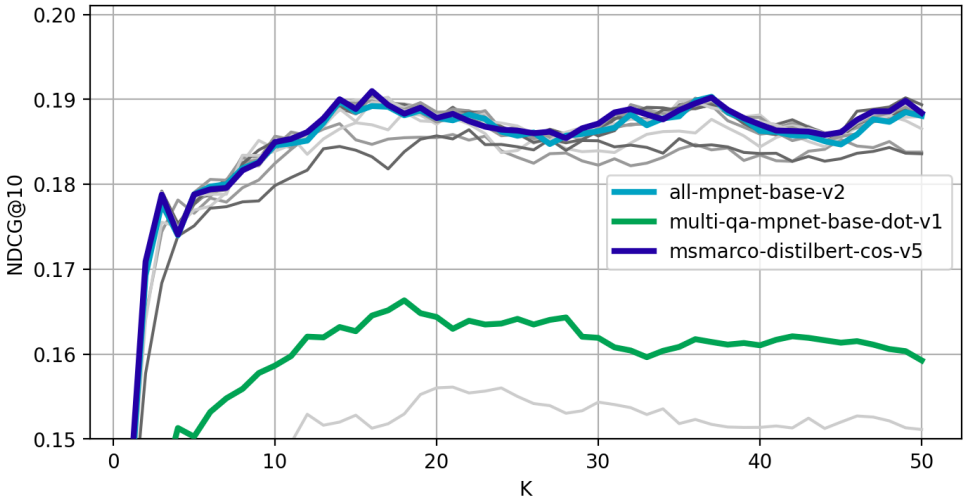

Figure 6.5. Nested evaluation of Hybrid ItemKNN on ml-100k for all embedding models with plot and cosine similarity. Relevant embedding models are highlighted.

As for Hybrid EASE^R, we ran all combinations of 21 models, and 2 fields. Fortunately, EASE^R has no similarity options and only one hyperparmeter λ. We tried the

following values:

$$\lambda \in \{0.01, 0.1, 1, 10, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 600, 700, 800, 900, 1000, 1200, 1400, 1600, 1800, 2000, 2500, 3000, 3500, 4000, 4500, 5000\}$$

In figure 6.6, we can see the best performing embedding model is `multi-qa-distilbert-dot-v1` with $\lambda = 250$ on the title field. This is what we'll use for the final evaluation.

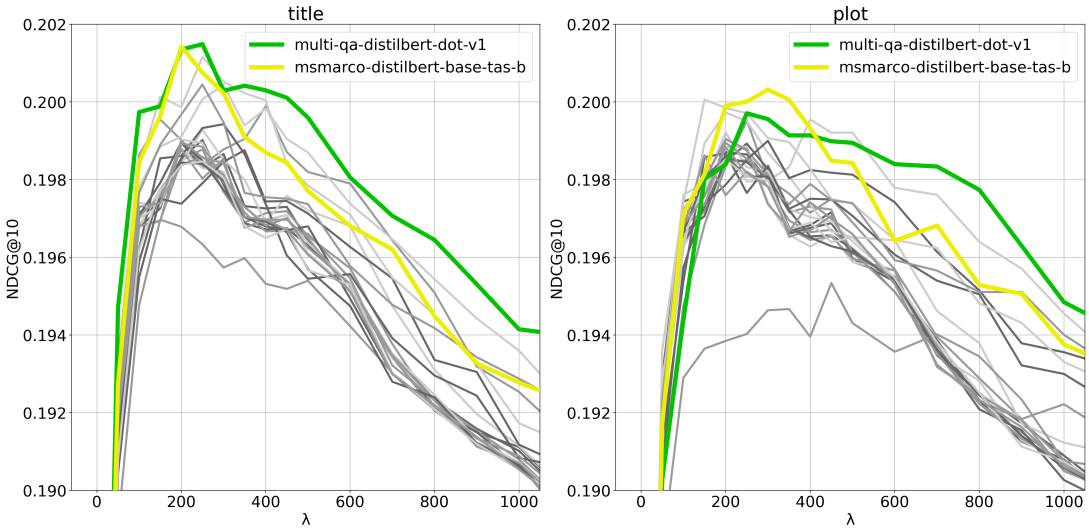

Figure 6.6. Nested evaluation of Hybrid EASE on ml-100k. Embedding models using the title field (left) and plot field (right). Values $\lambda > 1000$ are not displayed since the values are consistently lower and lower. Relevant embedding models are highlighted.

6.6 Results

See used hyperparameters and textual embedding models in table 6.4 for reference.

recommender	hyperparams	`textual embedding model`(field)
Top-Pop	—	—
EASE^R	$\lambda = 250$	—
UserKNN	$K = 20$, cosine	—
ItemKNN	$K = 14$, cosine	—
Content-based ItemKNN	$K = 20$, cosine	`multi-qa-mpnet-base-dot-v1`(title)
Hybrid EASE^R	$\lambda = 250$	`multi-qa-distilbert-dot-v1`(title)
Hybrid ItemKNN	$K = 16$, cosine	`msmarco-distilbert-cos-v5`(plot)

Table 6.4. Recommender hyperparameters for final evaluation on ml-100k. Textual embedding models aren't used in all methods.

Out of all of the baselines, EASE^R performs the best no matter the recommendation list length as can bee seen in table 6.5. It's important to note that the much older methods still perform not that far from EASE^R.

As for approaches with textual embeddings, Hybrid EASE^R performed the best for all recommendation list lengths over other approaches using textual embeddings. It also performed very slightly better than plain EASE^R but only for shorter list lengths. With longer list lengths, the performance is on par.

	@10				@20			
	HR	Prec.	Rec.	NDCG	HR	Prec.	Rec.	NDCG
Top-Pop	0.094	0.012	0.010	0.013	0.252	0.019	0.026	0.021
EASE^R	0.623	0.120	0.144	0.165	0.777	0.099	0.225	0.186
UserKNN	0.594	0.113	0.131	0.151	0.764	0.095	0.220	0.176
ItemKNN	0.597	0.105	0.130	0.148	0.747	0.087	0.202	0.167
Content-based ItemKNN	0.417	0.066	0.044	0.076	0.558	0.057	0.073	0.080
Hybrid EASE^R	0.629	0.123	0.145	0.167	0.781	0.099	0.229	0.187
Hybrid ItemKNN	0.594	0.105	0.133	0.147	0.738	0.087	0.203	0.166

	@50				@100			
	HR	Prec.	Rec.	NDCG	HR	Prec.	Rec.	NDCG
Top-Pop	0.477	0.022	0.066	0.044	0.608	0.019	0.109	0.069
EASE^R	0.930	0.075	0.398	0.246	0.972	0.057	0.554	0.302
UserKNN	0.889	0.070	0.373	0.230	0.952	0.053	0.518	0.282
ItemKNN	0.899	0.064	0.333	0.215	0.937	0.047	0.447	0.257
Content-based ItemKNN	0.728	0.043	0.132	0.099	0.855	0.034	0.209	0.128
Hybrid EASE^R	0.928	0.075	0.399	0.248	0.971	0.057	0.555	0.303
Hybrid ItemKNN	0.889	0.064	0.338	0.215	0.945	0.049	0.466	0.262

Table 6.5. Test set evaluation results for list length of $N \in \{10, 20, 50, 100\}$. Best result in each metric and N is underlined.

Chapter 7
Conclusion

We attempted to extend existing Top-N recommendation approaches with textual embeddings of items and evaluated the performance by time-splitting the data which should simulate real-world scenarios better, and used both item-presence and ranking metrics as recommended by [3]. In accordance with findings in [2], we used simple approaches for which we finetuned hyperparameters. The choice of the textual embedding model was individual to each approach where we tried all items' textual fields separately with a full range of hyperparameters for each method.

We managed to very slightly improve over baselines for shorter recommendation list lengths and kept performance on par for longer. However, why aren't the gains bigger? The embeddings definitely provide *some* information as demonstrated by semantic search and other NLP tasks [7]. Maybe we're not using them right?

Previous work focused mainly on using NLP approaches to user reviews and not items themselves which improved recommendation performance in [44]. Could somebody already have tried something similar, have found it not to work and not have published it?

Maybe the reason could be that the used dataset is too small or not suitable for this kind of approach. More experiments on more datasets could unveil the answer.

Another question is whether other kinds of content embeddings such as image embeddings could prove more useful for Top-N recommendation than textual embeddings. We leave this for future work. It's also possible that other recommendation tasks and scenarios benefit from textual embeddings of items better than Top-N recommendation.

References

[1] Covington, Paul, Jay Adams, and Emre Sargin. Deep Neural Networks for YouTube Recommendations. In: *Proceedings of the 10th ACM Conference on Recommender Systems*. New York, NY, USA: Association for Computing Machinery, 2016. pp. 191–198. RecSys '16. ISBN 9781450340359. Available from DOI 10.1145/2959100.2959190. Available from `https://doi.org/10.1145/2959100.2959190`.

[2] Dacrema, Maurizio Ferrari, Paolo Cremonesi, and Dietmar Jannach. Are We Really Making Much Progress? A Worrying Analysis of Recent Neural Recommendation Approaches. *CoRR*. 2019, Vol. abs/1907.06902. Available from `http://arxiv.org/abs/1907.06902`.

[3] Sun, Zhu, Hui Fang, Jie Yang, Xinghua Qu, Hongyang Liu, Di Yu, Yew-Soon Ong, and Jie Zhang. DaisyRec 2.0: Benchmarking Recommendation for Rigorous Evaluation. *arXiv preprint arXiv:2206.10848*. 2022. Available from `https://doi.org/10.48550/arXiv.2206.10848`.

[4] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In: I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017. Available from `https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

[5] Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*.

[6] Brown, Tom B., Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. *Language Models are Few-Shot Learners*.

[7] Reimers, Nils, and Iryna Gurevych. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*.

[8] Ricci, F., L. Rokach, and B. Shapira. *Recommender Systems Handbook*. Springer US, 2022. ISBN 9781071621967. Available from `https://books.google.cz/books?id=V6KzzgEACAAJ`.

[9] Ning, Xia, and George Karypis. SLIM: Sparse Linear Methods for Top-N Recommender Systems. In: *2011 IEEE 11th International Conference on Data Mining*. 2011. pp. 497-506. Available from DOI 10.1109/ICDM.2011.134.

[10] Deshpande, Mukund, and George Karypis. Item-Based Top-N Recommendation Algorithms. *ACM Trans. Inf. Syst.* New York, NY, USA: Association for Computing Machinery, jan, 2004, Vol. 22, No. 1, pp. 143–177. ISSN 1046-8188. Available from DOI 10.1145/963770.963776. Available from `https://doi.org/10.1145/963770.963776`.

[11] Recsys-Benchmark. *Recsys-benchmark/DAISYREC-v2.0*. Available from `https://github.com/recsys-benchmark/DaisyRec-v2.0`.

[12] Vančura, Vojtěch, Rodrigo Alves, Petr Kasalický, and Pavel Kordík. Scalable Linear Shallow Autoencoder for Collaborative Filtering. In: *Proceedings of the 16th ACM Conference on Recommender Systems*. New York, NY, USA: Association for Computing Machinery, 2022. pp. 604–609. RecSys '22. ISBN 9781450392785. Available from DOI 10.1145/3523227.3551482. Available from `https://doi.org/10.1145/3523227.3551482`.

[13] Koren, Yehuda, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*. IEEE, 2009, Vol. 42, No. 8, pp. 30–37.

[14] Rendle, Steffen, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. *BPR: Bayesian Personalized Ranking from Implicit Feedback*. Available from `https://doi.org/10.48550/arXiv.1205.2618`.

[15] Kang, Zhao, Chong Peng, and Qiang Cheng. Top-n recommender system via matrix completion. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2016.

[16] Rendle, Steffen. Factorization machines. In: *2010 IEEE International conference on data mining*. 2010. pp. 995–1000.

[17] Liang, Dawen, Rahul G. Krishnan, Matthew D. Hoffman, and Tony Jebara. *Variational Autoencoders for Collaborative Filtering*. Available from `https://doi.org/10.48550/arXiv.1802.05814`.

[18] Steck, Harald. Embarrassingly Shallow Autoencoders for Sparse Data. In: *The World Wide Web Conference*. New York, NY, USA: Association for Computing Machinery, 2019. pp. 3251–3257. WWW '19. ISBN 9781450366748. Available from DOI 10.1145/3308558.3313710. Available from `https://doi.org/10.1145/3308558.3313710`.

[19] He, Xiangnan, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural Collaborative Filtering. In: *Proceedings of the 26th International Conference on World Wide Web*. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2017. pp. 173–182. WWW '17. ISBN 9781450349130. Available from DOI 10.1145/3038912.3052569. Available from `https://doi.org/10.1145/3038912.3052569`.

[20] He, Xiangnan, and Tat-Seng Chua. Neural factorization machines for sparse predictive analytics. In: *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. 2017. pp. 355–364.

[21] Resnick, Paul, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In: *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*. New York, NY, USA: Association for Computing Machinery, 1994. pp. 175–186. CSCW '94. ISBN 0897916891. Available from DOI 10.1145/192844.192905. Available from `https://doi.org/10.1145/192844.192905`.

[22] Wang, Jun, Arjen P. de Vries, and Marcel J. T. Reinders. Unifying User-Based and Item-Based Collaborative Filtering Approaches by Similarity Fusion. In: *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York, NY, USA: Association for Computing Machinery, 2006. pp. 501–508. SIGIR '06. ISBN 1595933697. Available from DOI 10.1145/1148170.1148257. Available from `https://doi.org/10.1145/1148170.1148257`.

[23] Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space*.

[24] Pennington, Jeffrey, Richard Socher, and Christopher Manning. GloVe: Global Vectors for Word Representation. In: Alessandro Moschitti, Bo Pang, and Walter Daelemans, eds. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014. pp. 1532–1543. Available from DOI 10.3115/v1/D14-1162. Available from `https://aclanthology.org/D14-1162`.

[25] Arora, Sanjeev, Yingyu Liang, and Tengyu Ma. A Simple but Tough-to-Beat Baseline for Sentence Embeddings. In: *International Conference on Learning Representations*. 2017. Available from `https://openreview.net/forum?id=SyK00v5xx`.

[26] Kenter, Tom, Alexey Borisov, and Maarten de Rijke. Siamese CBOW: Optimizing Word Embeddings for Sentence Representations. In: Katrin Erk, and Noah A. Smith, eds. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, 2016. pp. 941–951. Available from DOI 10.18653/v1/P16-1089. Available from `https://aclanthology.org/P16-1089`.

[27] Pagliardini, Matteo, Prakhar Gupta, and Martin Jaggi. Unsupervised Learning of Sentence Embeddings Using Compositional n-Gram Features. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Association for Computational Linguistics, 2018. Available from DOI 10.18653/v1/n18-1049. Available from `http://dx.doi.org/10.18653/v1/N18-1049`.

[28] Dai, Andrew M., and Quoc V. Le. *Semi-supervised Sequence Learning*.

[29] Peters, Matthew E., Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. *Deep contextualized word representations*.

[30] Radford, Alec, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018 .

[31] Howard, Jeremy, and Sebastian Ruder. *Universal Language Model Fine-tuning for Text Classification*.

[32] Bowman, Samuel R., Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In: Lluís Màrquez, Chris Callison-Burch, and Jian Su, eds. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, 2015. pp. 632–642. Available from DOI 10.18653/v1/D15-1075. Available from `https://aclanthology.org/D15-1075`.

[33] Conneau, Alexis, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. In: Martha Palmer, Rebecca Hwa, and Sebastian Riedel, eds. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, 2017. pp. 670–680. Available from DOI 10.18653/v1/D17-1070. Available from `https://aclanthology.org/D17-1070`.

[34] Cer, Daniel, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. *Universal Sentence Encoder*.

[35] Sanh, Victor, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*. 2019 .

[36] Liu, Yinhan, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*.

[37] Song, Kaitao, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. *MPNet: Masked and Permuted Pre-training for Language Understanding*.

[38] Reimers, Nils, and Iryna Gurevych. Making Monolingual Sentence Embeddings Multilingual using Knowledge Distillation. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2020. Available from `https://arxiv.org/abs/2004.09813`.

[39] Melville, Prem, Raymod J. Mooney, and Ramadass Nagarajan. Content-Boosted Collaborative Filtering for Improved Recommendations. In: *Eighteenth National Conference on Artificial Intelligence*. USA: American Association for Artificial Intelligence, 2002. pp. 187–192. ISBN 0262511290.

[40] Musat, Claudiu, Yizhong Liang, and Boi Faltings. Recommendation using textual opinions. In: 2013. pp. 2684-2690.

[41] Ling, Guang, Michael R. Lyu, and Irwin King. Ratings Meet Reviews, a Combined Approach to Recommend. In: *Proceedings of the 8th ACM Conference on Recommender Systems*. New York, NY, USA: Association for Computing Machinery, 2014. pp. 105–112. RecSys '14. ISBN 9781450326681. Available from DOI 10.1145/2645710.2645728. Available from `https://doi.org/10.1145/2645710.2645728`.

[42] Seo, Sungyong, Jing Huang, Hao Yang, and Yan Liu. Representation Learning of Users and Items for Review Rating Prediction Using Attention-based Convolutional Neural Network. In: 2017. Available from `https://api.semanticscholar.org/CorpusID:38864450`.

[43] Zheng, Lei, Vahid Noroozi, and Philip S. Yu. *Joint Deep Modeling of Users and Items Using Reviews for Recommendation*.

[44] Ghasemi, Negin, and Saeedeh Momtazi. Neural text similarity of user reviews for improving collaborative filtering recommender systems. *Electronic Commerce Research and Applications*. 2021, Vol. 45, pp. 101019. ISSN 1567-4223. Available from DOI https://doi.org/10.1016/j.elerap.2020.101019. Available from `https://www.sciencedirect.com/science/article/pii/S156742232030096X`.

[45] Lops, Pasquale, Marco de Gemmis, and Giovanni Semeraro. Content-based Recommender Systems: State of the Art and Trends. In: Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, eds. *Recommender Systems Handbook*. Boston, MA: Springer US, 2011. pp. 73–105. ISBN 978-0-387-85820-3. Available from DOI 10.1007/978-0-387-85820-3_3. Available from `https://doi.org/10.1007/978-0-387-85820-3_3`.

[46] Li, Xiaopeng, and James She. Collaborative Variational Autoencoder for Recommender Systems. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2017. pp. 305–314. KDD '17. ISBN 9781450348874. Available from DOI 10.1145/3097983.3098077. Available from `https://doi.org/10.1145/3097983.3098077`.

[47] Zheng, Lei, Chun-Ta Lu, Fei Jiang, Jiawei Zhang, and Philip S. Yu. Spectral collaborative filtering. In: *Proceedings of the 12th ACM Conference on Recommender Systems*. ACM, 2018. RecSys '18. Available from DOI 10.1145/3240323.3240343. Available from `http://dx.doi.org/10.1145/3240323.3240343`.

[48] Gebremeskel, Gebrekirstos G., and Arjen P. de Vries. Recommender Systems Evaluations : Offline, Online, Time and A/A Test. In: *Conference and Labs of the Evaluation Forum*. 2016. Available from `https://api.semanticscholar.org/CorpusID:15506824`.

[49] Järvelin, Kalervo, and Jaana Kekäläinen. Cumulated Gain-Based Evaluation of IR Techniques. *ACM Trans. Inf. Syst.* New York, NY, USA: Association for Computing Machinery, oct, 2002, Vol. 20, No. 4, pp. 422–446. ISSN 1046-8188. Available from DOI 10.1145/582415.582418. Available from `https://doi.org/10.1145/582415.582418`.

[50] Harper, F. Maxwell, and Joseph A. Konstan. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* New York, NY, USA: Association for Computing Machinery, dec, 2015, Vol. 5, No. 4. ISSN 2160-6455. Available from DOI 10.1145/2827872. Available from `https://doi.org/10.1145/2827872`.