Web Recommender Systems Project

Daniel Penchev
University of Copenhagen
bqr255@alumni.ku.dk

1 Dataset analysis (Exercise 1)

A detailed analysis was conducted on the New Mexico partition of Google Local Data (2021) dataset. The dataset, which was split using a temporal approach, consists of 90000 item ratings provided by 1105 unique users on 666 unique items, with 80% of the data allocated to the training set and 20% to the test set.

Both the training and test sets were preprocessed by removing all ratings that lacked a user ID, item ID, or timestamp. Additionally, duplicate ratings with respect to the user ID and item ID were filtered out, retaining only the most recent rating among each group of duplicates. As a result, no samples were removed due to missing values, while 8992 ratings were removed from the training set and 5369 from the test set due to being duplicates. After the preprocessing step, the training set contained 61008 samples, while the test set contained 14631 samples. After this step, the sparsity of the whole dataset was found to be 89.7% (considering both the training and test sets).

The dataset also exhibits a strong positive bias in rating distribution (Figure 1), where the average rating for items is strictly above 3.0 (on a scale from 1 to 5) for each item in the training dataset. This observation either reinforces the idea that users tend to provide favorable ratings or suggests that during the dataset creation low-rated items were explicitly removed by mistake or on purpose. As a consequence, this rating skewness might lead to inflated rating predictions.

A further analysis of item popularity (Figure 2) reveals a long-tail distribution, where a small subset of items receives a disproportionately high number of high ratings (rating ≥ 3), while the majority of items are significantly less reviewed (≤ 80). This suggests that when choosing recommendation models, this skewness in the popularity should be taken into consideration, for instance, when calculating similarities between users. Surprisingly, according to (Table 1), the top 5 items with the highest reviews have received around 40-70 reviews, which places them around the end of the right tail of the distribution in (Figure 2). Due to having low number of reviews, these ratings might be considered unreliable.

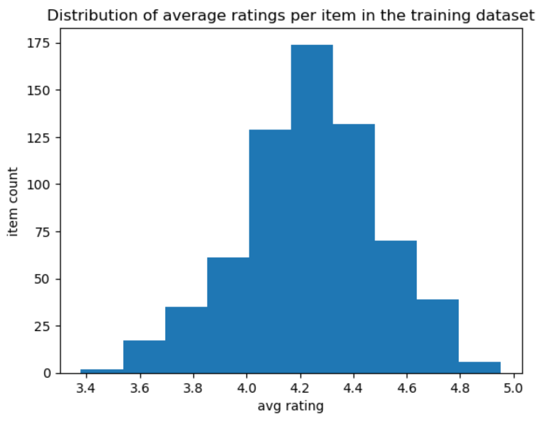

Figure 1: Distribution of average item rating in the training dataset.

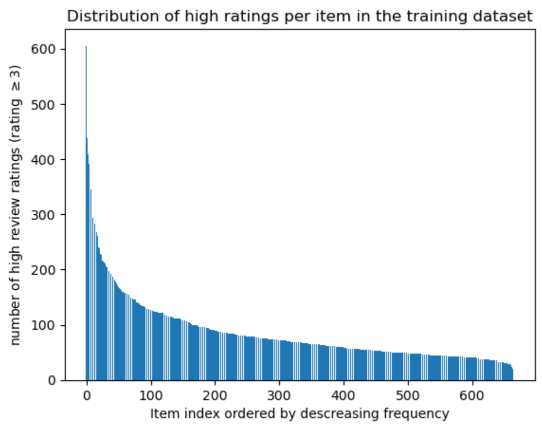

Figure 2: Distribution of high ratings per item in the training dataset.

Item id	average rate	number of rates
0x871877255c288679:0x48bf59554f723fd7	4.953125	64
0x86e06303071072f3:0x36d1188767bf20c3	4.916667	60
0x871830fbd3debbd9:0x4b42bb0128441324	4.857143	49
0x86e37913c4b8d75b:0x6ed43df7bd3bb2bd	4.850000	40
0x87220b36739a31b3:0x43c4a7597d6351e8	4.833	42

Table 1: Top-5 most highly-rated items in the train dataset.

These findings highlight several key considerations for recommendation models. The relatively high sparsity of the dataset may limit the effectiveness of collaborative filtering approaches, requiring models that can resolve the cold-start problem. On the other hand, the positive rating bias could lead to skewed predictions because only very few items have received individual ratings below 3, hence in our analysis only ratings ≥ 4 will be considered relevant. Finally, the presence of the temporal split of the data highlights the need for models that can adapt to changing user preferences over time.

2 Baseline Models (Exercise 2)

In this exercise, two non-personalized baseline models were implemented, *TopPop* and *RandomRec*, in order to establish performance benchmarks for the more advanced recommendation systems developed in the later Exercises.

The former model generates a single, global recommendation list by ranking all items based on the number of high ratings (≥ 4) they received in the training data. On the other hand, later model recommends k different items chosen randomly from the the training set.

Due to being non-personalized in nature, both are computationally efficient, require no tuning, and can serve as benchmarks. While, *TopPop* often acts as a strong baseline, especially when the dataset suffers from high sparsity, *RandomRec* can be seen as a lower-bound on the performance that any meaningful model must exceed.

3 Collaborative Filtering Recom- mender System (Exercise 3)

In comparison to Exercise 2, here the focus shifted towards exploring more advanced recommendation systems - such as latent matrix factorization and neighbourhood based models. The implementation of those models used Scikit-surprise library with seed 1234. For the hyperparameter tuning of those models, a **5-fold temporal-split cross-validation**(2) approach was used with **Mean Reciprocal Rank** with 10 cutoff as a selection metric (increased from 3 folds to improve stability). The ordinary KFold-CV procedure involves random splits, which cannot be used here, because it would introduce data leakage about the future in the training folds.

Mean Reciprocal Rank (MRR) is a rank-based metric that measures whether a relevant items is contained in a user-specific top-k recommendation list. Due to its simplicity it is computationally efficient, however it comes at a cost - it isn't particularly suited when there are multiple relevant items in the recommendation list. In comparison to its simpler version - **HitRate**, *MRR* actually considers the exact position of the item within the recommendation list.

For the next two types of models grid search was used for hyperparameter-tuning, exploring all value combinations of their hyperparameters. The hyperparameters and the their search space is specified in the Appendix 9.1.

The first personalized-model that was covered was *KNN*, a neighborhood-based collaborative filtering model that internally uses the K-nearest neighbors algorithm. It predicts a user's rating for an unobserved item by finding the k most similar users (user-based) or k most similar items (item-based). In the user-based approach, predictions are computed as a weighted average of ratings from similar users who have rated the item. While in the item-based approach, the model identifies similar items the user has rated and bases predictions on their ratings.

As a result of the hyperparameter-tuning procedure, the best overall performance was achieved with an item-based KNN (*MRR@10* of 0.055) using a single neighbor and *Pearson* similarity . However, using a single neighbour makes the model prone to overfitting to the training data, hence the second best set of hyperparameter values was used - item-based with 4 neighbours and Pearson similarity. This configuration achieved *MRR@10* of 0.054, which is extremely close to the best one, hence this tradeoff between potential performance and generalization performance can be justified. Moreover, the requirements of Exercise 8.1 necessitated training a user-based KNN model as well. This model was configured using the best hyperparameter value configuration that discovered for the user-based approach during the same tuning process instance. It achieved *MRR@10* of 0.053 with 4 neighbours and *Pearson* similarity.

The final personalized model that was experimented with was *SVD*, a model-based collaborative filtering approach, more specifically a latent factor model, that decomposes the user-item interaction matrix into a lower-dimensional representation. The main idea of the model is to capture the latent factors that explain user preferences and item characteristics. The predicted rating for a user-item pair is computed as the dot product of their respective latent factor vectors, adjusted by user and item biases.

As a result of the hyperparameter-tuning procedure it was determined that the best cross-validation performance was achieved when using 20 hidden factors and 50 epochs of SGD optimization. This configuration yielded an *MRR@10* of 0.054.

After the hyperparameter-tuning procedure, both personalized and non-personalized (Exercise 2) models were trained on the entire training dataset, and recommendations were generated for each user present in the training dataset.

4 Text Representation (Exercise 4)

In Exercise 4, it was explored how to represent text data so that it can be used by a different type of recommender systems - content-based. For this purpose only the train and test item descriptions in the metadata dataset were used, which reduced the metadata entries from 4409 to 666. No entries were removed due to having no description or name and no duplicates with respect to the item_id were found.

For the purpose of a fair evaluation, the content-based model's item set was constrained to only those items present in the training and test datasets. Using all available metadata samples would would have unfairly penalized the content-based model, as recommending any such item would have artificially degrade its performance metrics. This choice ensures a level playing field for comparing its performance against other models whose scope is inherently limited to the training/test items.

For the two model that we will experiment with in this exercise, the same metadata preprocessing steps were executed. Firstly, the descriptions and names were independently preprocessed, which included lowercasing, tokenization and removing stopwords, punctuation, and non-alphabetic tokens (except hyphenated words). In addition to that, numbers and words with punctuation were removed, due to containing little useful information. Finally, each the names and descriptions were independently converted to embeddings and concatenated. As a result of the preprocessing, the vocabulary, build from the names, was reduced from 731 to 658 words. In addition to that, the vocabulary, build from the descriptions, was reduced from 1405 to 1289 words.

Two ways of item metadata representations were examined - item representation using *Word2Vec* embeddings and item representation using *GloVe* embeddings. Due to both models being trained on full non-stemmed forms, they expect to receive their inputs in that same format at inference.

4.1 Word2Vec

Each word was represented using the *Word2Vec* model, which was trained on the ”Google News” corpus and whose vocabulary comprises more than 3 million distinct words, each mapped to a 300-dimensional embedding. For each item, its name and description were transformed to their vector representation by taking element-wise mean of the embeddings of the words is was comprised of, respectively. The final vector representation/embedding of each item had a dimensionality 600. The reason, why we use concatenation instead of just merging the two pieces of text into a single one and then converting it to a vector representation, was that the longer the text is, the more diluted the vector representation becomes due to the mean operation.

4.2 GloVe

A embedding-model which is very similar to *Word2Vec* is *GloVe* (3). While the former assumes words that appear in similar contexts have similar meanings, the latter assumes that the ratio of co-occurrence probabilities between words is the key to their meaning. A pre-trained GloVe model was chosen, which was trained on the ”*Wikipedia*

	0x34	0x78	0x79	0x30	0x24
0x34	1.00	0.57	0.27	0.30	0.13
0x78	0.57	1.00	0.23	0.24	0.12
0x79	0.27	0.23	1.00	0.69	0.13
0x30	0.30	0.24	0.69	1.00	0.11
0x24	0.13	0.12	0.13	0.11	1.00

Table 2: Cosine Similarity based on *Word2Vec* embeddings. The item_ids of the items were shortened by taking the first 4 symbols after the semicolon symbol.

(2014)" and "*Gigaword 5*" corpuses and whose vocabulary comprises more than 400K distinct words, each mapped to a 300-dimensional embedding. Item descriptions were pre-processed exactly as in the *Word2Vec* case. As before, the vector representation for each description was then obtained by taking the element-wise mean of the embeddings of the words is was comprised of.

	0x34	0x78	0x79	0x30	0x24
0x34	1.00	0.59	0.12	0.13	0.29
0x78	0.59	1.00	0.05	0.01	0.26
0x79	0.12	0.05	1.00	0.75	0.16
0x30	0.13	0.01	0.75	1.00	0.14
0x24	0.29	0.26	0.16	0.14	1.00

Table 3: Cosine Similarity based on *GloVe* embeddings. The item_ids of the items were shortened by taking the first 4 symbols after the semicolon symbol.

4.3 Similarity performance comparison

For each representation method (*Word2Vec*, *GloVe*), cosine similarities were computed among five items - two museums ('0x87227339ff929717:0x3422514fbdbffc0', '0x87220cd2b519870f:0x780df887461461f6'), two steakhouses ('0x87220ad3aaeabf87:0x79d5bcd7ff1266dd', '0x872274efda7f0051:0x30146b7f3460f963') and one hypermarket ('0x8722717d2d4eb66d:0x246f16d0e5892e05') and , in order to assess their quality (see Tables 2, 3).

According to the results, both *Word2Vec* and *GloVe* demonstrated strong performance, correctly identifying the high similarity between the two museums and between the two steakhouses.

However, the models differed in their treatment of the outlier item. The *Word2Vec* representation positioned the hypermarket as equally dissimilar to all other items. Conversely, the *GloVe* approach introduced a notable bias, linking the hypermarket more closely to the museums than to the steakhouses. While both models perform similarly at first glance, their different training methods can cause unexpected results in less straightforward comparisons. A further explanation for this discrepancy could be lie in vocabulary size difference. The chosen pre-trained *Word2Vec* model's larger vocabulary enables it to generate representations for a wider range of terms, potentially resulting in more complex understanding than *GloVe* pre-trained model. Hence, in Exercise 5 due to this particular reason we will be using *Word2Vec* instead of *GloVe*.

Note: Absolute similarity scores were not compared across representations, because each embedding space exhibits a distinct similarity distribution and scale, making an absolute value comparison invalid.

5 Content-Based Recommender Sys- tem (Exercise 5)

In this exercise a content-based recommender system (*CBRS*) with pre-trained word-embedding model *Word2Vec* (*W2V-CBRS*) was build using the metadata dataset as in the previous exercise.

Cosine similarity was preferred over *Euclidean* distance due to having to work with high-dimensional, as it focuses on direction rather than magnitude. *Euclidean* distance can wrongly identify similar items due to differences in length or term repetition.

Given our choice of item representation, the same transformation (as in the previous exercise) from item metadata to vector representation/item embedding was used. One of the drawbacks of concatenation the vector embeddings is that the dimension of the item embedding is doubled, but luckily it can be reduced using methods such as PCA, though this is left for future work. User profiles were computed using a weighted average of item embeddings based on mean-centered user ratings for the items the particular user has rated in the training dataset, and recommendations were generated by finding the most similar items to each user profile.

6 Hybrid Recommender Systems (Exercise 6)

Unfortunately, a *W2V-CBRS* only considers an item's metadata, while collaborative filtering recommender systems rely solely on user-item interactions. One way to leverage the strengths of both approaches is to use a hybrid strategy, where recommendations from different models are used for the creation of a single list. Three types of hybrid strategies were reviewed - parallel, switching and pipelining.

Parallel combination strategy (RRF): This type of aggregation function is used to combine the results recommendation lists of the *user-based KNN* and *W2V-CBRS* model. The core idea behind this strategy is to heavily favor items that appear moderately high up on multiple lists, as their combined reciprocal rank scores will quickly surpass items that only appear very high on a single list. One advantage of *RRF* over methods like *Round Robin* is that it weights each item candidate both by how many ranking sources include it and by how high it appears within each source, so top-ranked items from multiple lists rise to the top of the combined recommendation list. To combine recommendation lists, *RRF* was used with k set to 60.

Switching strategy (Switch): This approach selects the recommendation model based on how many times a user has rated items in the training set. If a user's count is below the highest count of the bottom 10% of users with least ratings, the *W2V-CBRS* model is used; otherwise, the *user-based KNN* model is chosen. In that way meaningful recommendations can be generated for cold-start users.

Pipelining strategy (Pipe): First, the *user-based KNN* model generates a candidate set of the top 10 items for each user, leveraging the collaborative signals. These candidates are then re-ranked by the *W2V-CBRS* model, which refines the list by scoring items based on their semantic similarity to the user's taste profile, therefore improving the personalization of the recommendations.

7 Evaluation of Recommender Systems (Exercise 7)

7.1 Comparison metrics

To provide a more comprehensive performance evaluation, we expand upon the Mean Reciprocal Rank (MRR) metric, introduced in Exercise 3, with the following new metrics:

Coverage measures the proportion of items in the catalog that are recommended to users at least once. It reflects how diverse a recommender system is in utilizing the available items. High coverage indicates that the model doesn't suffer from popularity bias. On the other hand, if this is the primary metric used for selecting the values of the hyperparameters during cross-validation, the process might actually select a relatively bad performing model compared to the best possible model.

Hit@k (averaged over all users): A simpler version of *MRR@k*. Like *MRR@k*, it is specifically suited for implicit feedback data or for scenarios in which there is only a single relevant item per user and measures whether that item is contained in user-specific top-k recommendation list. However, its biggest disadvantage, compared to *MRR@k*, is that it doesn't also take into account the position of the relevant item within the list.

Precision@k metric is a utility-based metric that counts the number of relevant items in the top-k recommendation list. However, unlike *MRR@k*, it ignores the exact positions of the relevant items within the list. Moreover, it also doesn't take into consideration whether all relevant items were included in the recommendation list or the exact degree to which a particular item is relevant (binary relevance). As a result *Precision* is well suited for evaluation of recommendation systems where the primary concern is the proportion of relevant items and not the ranking quality.

MAP@k is a utility-based metric that builds upon *Precision* by implicitly taking into account the position of the relevant items within a top-k recommendation list for each user, thereby heavily penalizing relevant items that appear lower in the list. Unlike *Precision*, *MAP* actually takes into consideration the true number of relevant items for a particular user.

7.2 Model comparison

Based on the evaluation results, the most surprising finding was the superior performance of the *TopPop* model across all ranking metrics, excluding *Coverage*. This dominance suggests that user behavior in this dataset is heavily concentrated on a very small set of popular items. The expected trade-off for this high accuracy is *TopPop*'s extremely low catalog coverage. Conversely, all personalized and hybrid models significantly outperformed the *RandomRec*

Model	Precision@10	MAP@10	MRR@10	Hit@10	Coverage@10
TopPop	0.052	0.025	0.140	0.321	0.015
RandomRec	0.018	0.007	0.045	0.157	0.998
User-based KNN	0.030	0.011	0.072	0.241	0.757
Item-based KNN	0.033	0.013	0.082	0.256	0.668
SVD	0.029	0.012	0.075	0.222	0.745
W2V-CBRS	0.028	0.013	0.076	0.211	0.919
RRF-HR	0.030	0.011	0.075	0.230	0.872
Switch-HR	0.031	0.013	0.078	0.239	0.860
Pipe-HR	0.033	0.012	0.083	0.256	0.668

Table 4: Performance evaluation of all models, using the testing dataset. For the computation of all rank-based measurement, all items, with average rating $\geq$ 4, were considered relevant.

baseline, confirming that they successfully learned meaningful patterns from the training data.

Among the collaborative filtering approaches, *item-based KNN* is the clear winner, outperforming both *user-based KNN* and *SVD* on all metrics except *Coverage*. The superiority of *item-based KNN* tells us that item-to-item relationships are more predictive than user neighborhoods in this dataset. On the other hand, *user-based KNN* delivered more diverse recommendations, reflected in its higher coverage. This suggests that might be less prone to suffer from the popularity-bias problem. *SVD*'s underperformance is indicative that its assumption of linear relationships might not be able to handle the data's complexity, making neighborhood-based methods more effective at capturing the relevant user-item interactions.

The content-based model, *W2V-CBRS*, showscases a typical trade-off between rank-performance and diversity of its recommendations. While its ranking metrics are lower than *item-based KNN*, it achieved the second-highest catalog coverage. This is a key strength of content-based systems, which can recommend any item with metadata, thus not being influenced by the popularity bias that affects collaborative models. However, its performance is inherently limited to recommending items similar to what user already rated, preventing it from discovering new connections that collaborative models can identify.

Surprisingly, none of the hybrid recommender models managed to outperform the *item-based KNN* model. The hybrid approach with pipelining, which re-ranks *item-based KNN*'s recommendations, came the closest. The results reveal an interesting trade-off - re-ranking slightly improved *MRR@10*, suggesting it successfully promoted some relevant items in particular situations, but it degraded *MAP@10*, indicating it worsened the overall ranking of other relevant items in the list. The underperformance of the hybrid model with *RRF* merging strategy isn't surprising. The fact that on average only two items were common between the component models' candidate lists means the *RRF* strategy had no meaningful overlap to work with, effectively being equivalent to a simple Round-Robin strategy. This highlights that the *RRF* strategy is only effective when the models provide overlapping sets of high-quality candidates.

8 Explanations for Recommender Systems (Exercise 8)

8.1 Explanation study on User-based KNN's recommendations

To analyze the performance of the *user-based KNN* model on a deeper level, 3 users were selected based on two criterion - their Reciprocal Rank (RR@10) and the number of their top-10 globally closest neighbors who had actually rated the top-1 item recommended to them. The chosen users are:

- ”Ordinary User”: A user with RR@10 of 1.0, whose top recommended item has been rated by 6 of his 10 globally closest neighbors. (user id 1.0558910065723009e+20)
- ”Low RR Paradox User”: A user with RR@10 of 0.0, despite that 8 of his 10 globally closest neighbors had rated his top recommended item. (user id 1.0561129522668233e+20)
- ”High RR Paradox User”: A user with RR@10 of 1.0, even though none of his 10 globally closest neighbors had rated

the his top recommended item. (user id 1.011898710915715e+20)

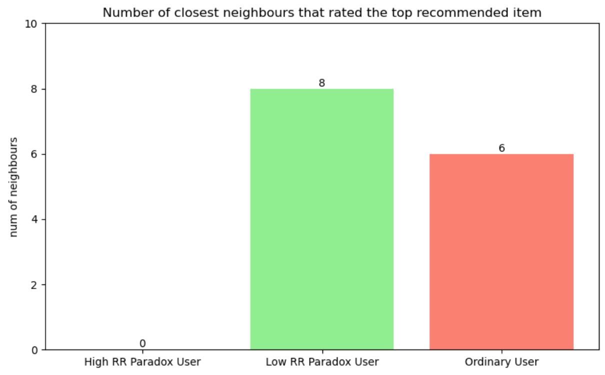

Figure 3: Number of a target user's top 10 closest neighbors who have rated the top item recommended for that user.

The model's vulnerability to popularity bias is evident as both the ”Ordinary User” and ”Low RR Paradox User” received the same top-1 recommendation (”0x87220cc2cc1c1055:0x52cc653371258502”). This item is highly popular, rated by a significant portion of the user base (437 out of 1105), suggesting its selection was driven by global popularity rather than individual user preference. This finding suggests the existence of a popularity bias problem with the *user-based KNN* that also can be observed in Figure 3.

On the other hand, the ”High RR Paradox User” has been recommended ”0x87189ff270f8c8bf:0x6801529028272065”, which can be seen as a niche item since 36 users rated it. This item's top position can be attributed to a predicted rating of 5.0 from our *user-based KNN* model (k=4). This prediction was calculated from the ratings of the four most similar users who had reviewed the item, three of whom assigned it the maximal score of 5.0. Moreover, with 6 items tied for the top predicted score of 5.0, the niche item's number-one position was decided by the model's tie-breaking implementation.

Interestingly, 19 out of 27 items, rated highly by that user (≥ 4.0), are actually food related businesses. Moreover, there is a huge bias in the training dataset itself towards rating such businesses. Out of 665 items in the training dataset, 433 were with an avg rating ≥ 4.0 and 86% of those items were precisely food related businesses, which explains why the top 1 recommended item was a food related one.

8.2 Ablation study on W2V-CBRS's features

Model	Precision@10	Hit@10
Without name	0.067	0.333
Without description	0.033	0.333
Using all features	0.033	0.333

Table 5: Ablation study on the performance of the W2V-CBRS model using the 3 users from section 8.1.

In this subsection we will analyze the outcomes of the ablation study that was conducted on the *W2V-CBRS* model. Similarly to Exercise 5, all metadata features except description and name were ignored

The study revealed (Table 5) that the description feature is the most informative feature (according to *Precision@10*), yielding a *Precision@10* twice as high as the name feature alone. This suggests a negative feature interaction where the weaker name feature pollutes the information from the description, harming the model's overall performance.

One possible explanation for this performance degradation might be due to the naive concatenation of feature embeddings, which gives equal weight to all vector dimensions. The reason why the name feature is noisy is due to many names lacking key words (for instance ”cabela” instead of ”cabela retail store”).

In addition to that, we can see the discrepancy between the values of *Hit@10* and *Precision@10*. A steady *Hit@10* of 0.333 indicates that one specific user always received a relevant recommendation, while the others did not. The drop in precision from 0.067 to 0.033 when the name feature was added reveals that relevant item/s was/were pushed out of the top-10 list for a user among the user used in this experiment. This directly demonstrates that the name feature harmed ranking quality, rather than simply failing to contribute.

However, it is critical to note the limitations of this small-scale ablation study. We cannot conclude that the description feature is generally twice as important as the name feature, because such a statement would require a more computationally intensive experiment (on all users with a broader set of rank-based metrics). While *Hit@10* is equally invariant to the ablation of the two features, the *Precision@10* shows that the description feature provides a better signal for ranking quality in this context.

9 Appendix

9.1 Exercise 3

In Exercise 3 two types of collaborative filtering methods were trained - neighbourhood-based (KNN) and matrix-factorization-based (SVD). For the selection of hyperparameter values a cross-validation procedure was employed. The search grids are described in Table 6 and Table 7.

Hyperparameter	value options
Nearest neighbours (K)	1, 2, ..., 33
Prediction type	user-based, item-based
Similarity measure	Cosine, Pearson

Table 6: Hyperparameters and their possible values that were used for tuning the KNN model.

Hyperparameter	value options
Hidden features/factors	2, 5, 10, 15, 20, 25, 30
Epochs (for SGD optim.)	20, 50, 100, 1000

Table 7: Hyperparameters and their possible values that were used for tuning the SVD model.

As it was noted before, during the hyperparameter tuning of the KNN model, it was found that using a single neighbour consistently was among the top configurations regardless of the prediction type (item-based or user-based). This peculiarity signalizes an aspect of our data, which should have been further investigated, but was left as future work.

9.2 Exercise 4

9.2.1 TF-IDF text embeddings

TF-IDF improves upon the Bag-of-Words model by accounting for word importance across the corpus. While BoW treats all words equally, TF-IDF reduces the influence of common terms and emphasizes rarer, more distinctive words, resulting in more meaningful text/document representations.

The preprocessing process mentioned in Exercise 4 was altered by using in addition a lemmatizer just before the removal of stop words and punctuation.

Instead of first transforming the values each of the two features (description and name) to an embedding and then concatenating them, here a simpler approach was employed. At the very beginning the description and the name texts are concatenated before the preprocessing step. This merged item text representation is directly transformed to a TF-IDF embedding.

	0x34	0x78	0x79	0x30	0x24
0x34	1.00	0.21	0.02	0.0	0.0
0x78	0.21	1.00	0.0	0.0	0.0
0x79	0.02	0.0	1.00	0.35	0.01
0x30	0.0	0.0	0.35	1.00	0.01
0x24	0.0	0.0	0.01	0.01	1.00

Table 8: Cosine Similarity based on *TF-IDF* embeddings. The item_ids of the items were shortened by taking the first 4 symbols after the semicolon symbol. The full ids of the items are described in Exercise 4

According to Table 8, the results are considerably different to the results of Word2Vec (Table 2), but still the museums were found to be the closest to one another, and so do the steakhouses. However, the difference is that some of the similarities were evaluated to be precisely 0.0. This can be explained by how tf-idf embeddings work - the text representations of two items, whose cosine similarity of their tf-idf vectors is 0.0, haven't got any words in common. Therefore, making their tf-idf vectors orthogonal. One of the most significant drawback of the tf-idf representation is that it doesn't learn the semantics of the item text representation, instead it treats similar words as being non-related.

9.2.2 BERT text embeddings

A significant limitation of the representation approaches - *TF-IDF* and word embeddings with *Word2Vec* - is that they map each word to a representation independently, ignoring context their were used in. In contrast, *BERT* can generate a contextual embedding that summarizes the entire item description. In order to use the *BERT* model, a dedicated *BERT* tokenizer with vocabulary size of 30522 words was used on raw descriptions, which in addition adds model-specific tokens and truncates long inputs. The full tokenized description is passed to *BERT*, and the embedding corresponding to the special [CLS] token is extracted as the summary representation.

According to Table 9, BERT, similarly to the approach with *Word2Vec*, mistakenly identified a discount store (”0x8722717d2d4eb66d:0x246f16d0e5892e05”) as the closest item to the steakhouse ’0x87220ad3aaeabf87:0x79d5bcd7ff1266dd’ (and the other way around). Not surprisingly, the

	0x34	0x78	0x79	0x30	0x24
0x34	1.00	0.86	0.85	0.72	0.81
0x78	0.86	1.00	0.84	0.79	0.80
0x79	0.85	0.84	1.00	0.83	0.88
0x30	0.72	0.79	0.83	1.00	0.80
0x24	0.81	0.80	0.88	0.80	1.00

Table 9: Cosine Similarity based on *BERT*'s [CLS] token embedding. The item_ids of the items were shortened by taking the first 4 symbols after the semicolon symbol. The full ids of the items are described in Exercise 4

vanilla *BERT* model doesn't outperform the other 2 methods due to being trained on a different objective (sentence continuity and masked word prediction).

9.2.3 Sentence BERT embeddings

As mentioned in the previous subsection, *BERT* (*vanilla BERT*) wasn't trained on text summarization tasks, hence the output corresponding to the [CLS] token might not necessarily contain a useful text summary. To overcome this problem, a fine-tuned BERT model on the text summarization task was used - *Sentence BERT (SBERT)* (1). This model directly outputs the similarities between text descriptions, by internally compressing the text format into an embedding of size 384 and computing the similarities between the embeddings.

	0x34	0x78	0x79	0x30	0x24
0x34	1.00	0.56	0.193	0.14	0.11
0x78	0.56	1.00	0.157	0.16	0.07
0x79	0.193	0.157	1.00	0.57	0.194
0x30	0.14	0.16	0.57	1.00	0.192
0x24	0.11	0.07	0.194	0.192	1.00

Table 10: Cosine Similarity based on *SBERT*'s text embedding. The item_ids of the items were shortened by taking the first 4 symbols after the semicolon symbol. The full ids of the items are described in Exercise 4

Comparing the relative results in Table 9 and Table 10, we can clearly see that the *SBERT* has a better understanding of how to summarize texts and compare their similarities. More concretely, vanilla *BERT* wrongly determined ”0x87220ad3aaeabf87:0x79d5bcd7ff1266dd” (steakhouse) to be closest to ”0x8722717d2d4eb66d:0x246f16d0e5892e05” (discount store) . In contrast, *SBERT* not only correctly recognized that ”0x872274efda7f0051:0x30146b7f3460f963” (steakhouse) is the closest item to the aforementioned steakhouse, but also the relative difference between the similarities is much higher than what we have observed with vanilla BERT. Moreover, it also recognized that there aren't any similar items to ”0x8722717d2d4eb66d:0x246f16d0e5892e05” (discount store), purely judging by the orders of magnitude lower similarity values with the other items compared to the self-similarity value.

9.3 Exercise 4/6

In addition to the models presented in Exercise 5 several other content-based models were examined, such as: TF-IDF Content-based recommender (TF-IDF-CBRS), GloVe Content-based recommender (GloVe-CBRS) and SBERT Content-based recommender (SBERT-CBRS).

Moreover, the suite of parallel hybrid models was also enriched with models such as *Round-Robin* Hybrid recommender, *CombMnz* Hybrid recommender, *CombMin* Hybrid recommender, *CombMax* Hybrid recommender and *CombMean* Hybrid recommender, where:

- the *CombMnz* strategy sums up the ratings of particular item in all input recommendation list and multiplies that number by the number of lists where the item occur.
- the *CombMax/CombMin* strategy involves taking the highest/lowest rating that was given to a particular item in any of the recommendation lists.
- the *CombMean* strategy entails taking the mean of the ratings for a particular item.

What is crucial as a input recommendation list preprocessing step for all of the aforementioned Hybrid recommender system was transforming all ratings to be on the same scale. Content-based recommendation systems create recommendations based on the cosine simularity, which is a value between 0 and 1, while the collaborative filtering systems work on the 1 to 5 scale. Hence, in our case it was decided to apply a Z-score normalization on each recommendation list, where the mean and standard deviation were globally computed for the particular list.

9.4 Exercise 7

Due to space constrains, the additional models mentioned in the previous section in the Appendix weren't analyzed in section about Exercise 7. The full model comparison information can be found in Table 11.

In section 7.2, when we reviewed the performance of the *RRF* hybrid recommendation model, it was suspected that its performance would be very similar to that of a hybrid model, which uses *Round Robin* as a merging strategy. This assumption is also confirmed by Table 11, where the *Round Robin* Hybrid model outperforms the RRF model with respect to all metrics, except Coverage.

What was also surprising is the performance of the *TF-IDF* Content-based recommender, which is considerably higher than all other models, except *TopPop*. Given how short the item names and descriptions were, it was assumed that approaches such as *TF-IDF* won't be able to perform well due to not taking the semantic meaning of the words into consideration. On the other hand, despite *SBERT* sentence embedding model being explicitly trained to compare the semantic similarities between two pieces of texts, it doesn't improve significantly the performance of the Content-based model when replacing the *Word2Vec* embedding model.

References

[1] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2019. Available at: https://arxiv.org/abs/1908.10084.

[2] Pedro G. Campos, Fernando Díez, and Iván Cantador. Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols. *User Modeling and User-Adapted Interaction*, 24(1–2):67–119, 2014. doi:10.1007/s11257-012-9136-x.

[3] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, 2014. Association for Computational Linguistics. Available at: https://aclanthology.org/D14-1162.

Model	**Precision@10**	**MAP@10**	**MRR@10**	**Hit@10**	**Coverage@10**
TopPop	0.052	0.025	0.140	0.321	0.015
RandomRec	0.018	0.007	0.045	0.157	0.998
User-based KNN	0.030	0.011	0.072	0.241	0.757
Item-based KNN	0.033	0.013	0.082	0.256	0.668
SVD	0.029	0.012	0.075	0.222	0.745
W2V-CBRS	0.028	0.013	0.076	0.211	0.919
TF-IDF-CBRS	0.038	0.019	0.102	0.276	0.817
GloVe-CBRS	0.030	0.014	0.083	0.224	0.914
SBERT-CBRS	0.030	0.014	0.075	0.211	0.925
RRF-HR	0.030	0.011	0.075	0.230	0.872
Round-Rob-HR	0.031	0.012	0.080	0.236	0.868
CombMean-HR	0.033	0.015	0.086	0.253	0.905
CombMin-HR	0.034	0.015	0.089	0.265	0.905
CombMax-HR	0.034	0.016	0.089	0.257	0.898
CombMnz-HR	0.033	0.015	0.084	0.252	0.907
Switch-HR	0.031	0.013	0.078	0.239	0.860
Pipe-HR	0.033	0.012	0.083	0.256	0.668

Table 11: Performance evaluation of all models, using the testing dataset. For the computation of all rank-based measurement, all items, with average rating ≥ 4, were considered relevant.