Web Recommender Systems

Armin Irvije
University of Copenhagen
rjc588@alumni.ku.dk

1 Week 6

I begin with data exploration and data processing. The dataset, which consists of 1389 unique users and 932 unique items, with 37,500 interactions split into a test and train set. The datasets have no rows containing empty cells. All users in the test split are also in the train split. 3420 and 855 duplicated rows were removed in the train and test sets, respectively. Taking the distribution of the frequency of ratings, we found that 5 makes up the most ratings given (16000), and 2 is the lowest (1500).

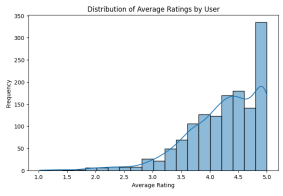

Figure 1: Distribution of ratings by users.

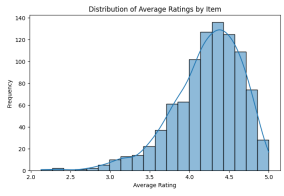

Figure 2: Distribution of ratings by items.

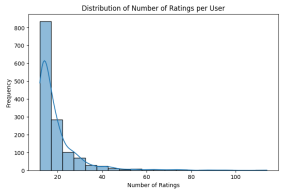

Figure 3: Number of ratings per user.

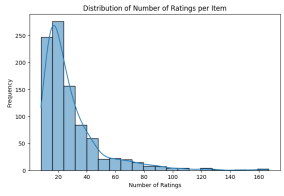

Figure 4: Number of ratings per item.

Figures 1 and 2 show the distribution of ratings by users and items, respectively. Both distributions are skewed right, with a large concentration of high ratings. We can observe that there are a few outliers. However, there is a clear positivity bias in the distribution of ratings by users. Most users rate items quite highly. This could skew models by allowing a naive model to simply predict a high rating. The results from root mean squared error would look deceptively good. Figures 3 and 4 show the number of ratings per user and per item, respectively. The distributions show a common characteristic of a long tail.

Most users contribute a small number of ratings, while a small subset accounts for a large share of the interactions. On the item side, a small number of items receive the majority of ratings and most items are rated a few times. This creates large data sparsity and pushes CF models to focus on frequent users and the most popular items. This results in strong popularity bias and low coverage of the long tail.

Item ID	Frequency	Average Rating
B0086VPUHI	154	4.43
B00BN5T30E	150	4.43
B07YBXFDYN	146	4.33
B00BGA9WK2	136	4.43
B007CM0K86	123	4.55

Table 1: Top 5 most frequently and highly rated items (Top Pop).

Next, I define 'highly rated' and 'relevant' as having a rating greater than or equal to 3. I compute the frequency of how many times an item was rated highly. Table 1 shows the 5 items with the most relevant rating sorted by frequency, along with their average rating. This TopPop is used as a baseline model and is the most primitive recommender system as it recommends every user the same global popular items (Maistro and Lioma, 2026). Lastly, I create user-item interaction matrices for the train and test split and calculate their sparsity. the train split has a sparsity value of 0.98, and test has a sparsity value of 0.99, showing that only 1-2% of possible interactions are observed. So many unobserved interactions make it difficult for certain models to be accurate in their predictions. Models will struggle to learn similarities between items and user patterns.

2 Week 7

This week I experimented with collaborative filtering (CF) models, which use other users' ratings and past behavior to make a prediction about what a user will like in the future. I trained an item-based K-nearest neighbors (KNN) model and a singular value decomposition (SVD) model. I ran a grid search with 5 folds on both models. Table 2 shows the configuration of hyperparameters trained on the grid search. I used mean absolute error (MAE) for my performance metrics because it provides a more general score on accuracy whereas root mean squared error (RMSE) penalizes outliers more heavily which our dataset does not have a lot of.Ranking metrics such as Precision@k and Mean Reciprocal Rank are also an important consideration, because they better reflect how accurate the top-k recommended list is for each user. While MAE summarizes overall rating error, it does not provide insight into how good the ranked lists are. In the following week, I evaluate the models on the test set using ranking metrics.

KNN is a memory-based CF because it directly uses the user-item matrix to make predictions. For the KNNBasic model I choose item-based because in a practical setting item-based scales better as a system grows with users. User-based KNN has to maintain similarities between users, thus when a commercial system grows fast through users joining, updating similarities between users is computationally expensive. Items in a commercial system grow more slowly which makes the computations of similarities more stable. It is also more explainable in a commercial system as its easy to say an item is recommended to a user because its similar to an item they interacted with. The best configuration was 5 nearest neighbors, 5 epochs, and cosine similarity with a MAE of 0.69. The SVD model is an approach that compresses the user-item matrix into dense embeddings, reducing the dimensionality to generate predictions. The best hyperparameter configuration was 25 latent factors, 50 epochs, 0.05 learning rate, and 0.05 regularization for overfitting. The MAE was 0.68. Both models achieve similar MAE likely due to the positivity bias, where simply predicting high ratings (4-5) can yield good performance. A model that predicts high ratings can achieve a low MAE. After choosing the best hyperparameter setup for each model, I predict ratings on all the missing entries in the user-item matrix.

Table 2: Hyperparameter grids used in 5-fold cross-validated grid search.

Model	Hyperparameter	Values Searched
KNNBasic	`num_neighbors`	{5, 10, 20, 25}
	`epochs`	{5, 10, 15, 20}
	`similarity_func`	{cosine, pearson}
SVD	`n_factors`	{25, 50, 100, 150, 175}
	`n_epochs`	{20, 30, 40, 50}
	`lr_all`	{0.002, 0.005, 0.01}
	`reg_all`	{0.02, 0.05, 0.1}

3 Week 8

This week, I created a top-10 recommendations list per user from the predictions on the train split. I evaluate the TopPop, KNN and SVD models on the test split. The evaluation metrics and their results are shown in Table 3. First, I evaluate using RMSE. The results were 0.98 for KNNBasic and 0.94 for SVD. This indicates that SVD makes slightly fewer errors in predicting ratings. RMSE only measures prediction values to ground truth not the quality of a top k recommendation list or the relevancy of recommendations. The metric is also sensitive to outliers and does not incorporate the sparsity issue. A good RMSE or MAE does not translate into a good ranking of top-k recommendations. For the ranking-based metrics I set $k = 10$ and evaluate on the quality of the top-10 recommendations list.

3.1 Hit Rate

Hit rate is a simple metric that calculates the number of times an item a user interacted with that is relevant appears in their top-k recommendation. We find the hit rate and sum them, and then divide by the total number of users. TopPop performs the best out of all the models. Roughly 21% of users' have at least one relevant test interaction in their top 10 list compared to only 1.6% for KNN and 5.5% for SVD. TopPop has more of the items included due to the popularity bias, meaning more users have interacted with at least one item in the TopPop list. A model recommending items at the head of a long tail distribution will hit users interactions much more. This rank does not incorporate relevancy.

3.2 Precision@k

Precision@10 calculates how many relevant items were included in the top-10 recommended items divided and then averaged by the total number of

users. All models perform quite low, with TopPop once again being the highest, but by a smaller margin. It provides a score for how well relevant items appear in the top 10 recommendation list. TopPop's precision reveals that only about 0.24 relevant items appear in the top 10. SVD roughly triples KNN's precision but both remain far below the popularity baseline. The low precision is expected because of the high sparsity in the test set.

3.3 Mean Average Precision

Mean Average Precision (MAP) is useful for checking how many relevant items are in the top 10 list and how early they appear in the list. All models have low scores with the baseline being the best once again. This means that relevant items are rare and appear low in the top-10 list. MAP gives more weight to relevant items appearing high in the recommendations list. The results show that even personalized models are not ranking relevant items high enough.

3.4 Mean Reciprocal Rank

Mean Reciprocal Rank (MRR) punishes the model for placing the first relevant item low on the list. MRR is a good metric for providing insight into the quality of the recommendations list. It shows how early a user sees a relevant item. KNN with an MRR of 0.005 means that there is no useful ordering of the recommended items.

3.5 Coverage

Coverage measures how many items in the catalog appear in the top 10 lists of all users' recommendations. This metric is useful if a company is interested in being diverse in its recommendations and in checking for bias towards certain items. KNN with a value of 0.89 indicates that almost the entire catalog appears in the recommendations, whereas TopPop uses 1% which is expected because it only shows one global popularity list. The trade-off is that higher coverage can mean lower accuracy in the predictions.

3.6 Model Comparison Analysis

Across all the metrics the common trend is that TopPop performs the best, then SVD, then KNN. The strong positivity bias and extreme sparsity result in the most interacted with items are also likely to appear in a user's test interaction. This gives TopPop an advantage in metrics like Hit Rate and Precision@10. In contrast it has super low coverage. TopPop recommends the popular items but does not provide anything personalized to users. KNN has very low metrics, but the highest coverage. While this diversity can be desirable it comes at a cost of poor accuracy. The low performance on all the other metrics indicates that the majority of its recommendations do not correspond to items actually interacted with in the test dataset. This is a consequence of CF's sensitivity to sparsity. SVD is a bit better than KNN but still worse than TopPop on rankings. SVD can generalize better from sparse interactions and learn the underlying patterns. But it still falls short of TopPop performance. This is likely due to not being sufficiently tuned and limited to the range of grid search hyperparameters. CF models spread their high predictions across more items, thus using their predictive ability on things that don't appear in the test set. The results show global popularity performs the strongest against the CF-trained models.

Table 3: Evaluation metrics for all models on the test set.

Model	HR	P@10	MAP@10	MRR	Cov.
KNN	0.016	0.0016	0.0009	0.005	**0.89**
SVD	0.0553	0.0057	0.0051	0.0180	0.4757
TopPop	**0.2133**	**0.0236**	**0.019**	**0.0709**	0.0107

3.7 Long Tail Effect

Model	Top 20%	Bottom 20%
User HR		
KNN	0.0289	0.0181
SVD	0.1155	0.0325
Item Coverage		
KNN	0.5161	1.0000
SVD	0.6828	0.3280

Table 4: Long-tail performance of KNN and SVD on users (hit rate) and items (coverage) for the top and bottom 20%.

I create two subsets of the training data, the top 20% of users and the bottom 20% of users in terms of how many ratings they provided. I Calculate hit rate based on the subsets for KNN and SVD. The Table 4 shows what is expected, that the highest users had a better hit rate then the bottom users. This showcases the cold start issue where users with less history provide less information for the models. For items we calculate coverage on the

top and bottom 20%. The results in SVD indicate popularity bias. But for KNN the head items had a coverage of 0.52 and bottom 20% had a coverage of 1.0, reflecting the super high overall coverage score it had. It also indicates that it recommends unpopular items which is why it performs bad in all other metrics.

3.8 Error Analysis for Nearest Neighbors

To investigate where item-based KNN model succeeds and fails, an item level reciprocal rank (RR) was computed. I found the highest RR item (RR = 1.0) and lowest RR item (RR=0.0). Both items had similar average rating in the test set, suggesting that the difference in RR is not because of item quality. Inspecting the 10 nearest neighbors of each reference item showed that all neighbors had a cosine similarity of 1.0 for both items. I also checked to see if there were overlapping users that rated the reference items and its neighbors. High RR had no overlap and low RR had just two. The primary issue for KNN is data sparsity. Items with few ratings cannot form meaningful similarity relationships cause the model retrieves neighbors that share no common user base. While coverage may be high this undermines ranking quality. For those items where item-based KNN performs poorly (≤ 0.05), the main issue is again the density of the data. As we saw in the data exploration many items have few ratings which lead to little overlap in the user sets between these items. As a result, cosine similarity is computed on almost no shared signal causing the model to retrieve nearest neighbors that look high numerically but do not share meaningful user base.

4 Week 9

In this week I used several NLP techniques to convert item descriptions text into numerical representations. Because machine learning models cannot be fed raw text, we need to convert them into numerical representations that models can run computations on. I focus on two main techniques: Term Frequency Inverse Document Frequency (TF-IDF) and pretrained word embedding model. This is relevant because item metadata and user preference is often described with natural language (description, genre, title, etc.). Even before we convert the text there are different methods of standardizing it such as lemmatization, tokenization, lower-casing, and stop-word removal. I decided to conduct text preprocessing on the subset of items that appear in train and test sets. This narrows the metadata space to only items that actually appear in the datasets and avoid spending computation cost on items that never appear in interactions. For each item, I used the description metadata column because it provides a short summary of the video game. The first phase of the preprocessing pipeline involves standardizing the text. I convert all characters to lower case and remove any non alphanumeric characters. Then I remove any stop-words which are repeated and common words such prepositions and pronouns. A key goal is to reduce noise and the vocabulary size which will make downstream tasks more computational efficient. I decided not to use lemmatization or stemming in my preprocessing. The descriptions in the dataset are already brief and simple, I did not see a reason to reduce words into their base or stem forms. I felt I could over normalize and lose the linguistic features of the descriptions. The initial vocabulary size went from 40,011 to 18,440. This reduction indicates that many uninformative characters were removed. I also removed rows with empty descriptions. Both TF-IDF vectorizer and embeddings models automatically tokenize into sub-words when applied on the text, meaning the library splits the description into subword unit before converting them into numerical vectors. To compare how well text-based features capture similarity between items, I computed cosine similarities on a random subset of 100 items from the test set. For each item in the subset I create TF-IDF representations of their descriptions and a pretrained embedding model. The word embedding model I used was `all-MiniLM-L6-v2` from HuggingFace (Wang et al., 2020). Table 5 displays the cosine similarity scores for least similar and most similar pairs with respect to their embedding method. As we see in the results, TF-IDF assigns the highest similarity scores to items whose titles share many exact tokens imply that their description also have very similar tokens. It also reduces to 0 for pairs with no shared tokens. This is because TF-IDF builds a vector where each unique word corresponds to a specific dimension. The value in that dimension is based on how often a word appears in a text with a saturation factor, and how rare the word is across the text. Therefore the similarity of TF-IDF depends on overlapping tokens. It has a hard time separating slight differences from very different. In contrast, word

Table 5: Representative cosine similarity pairs comparing TF-IDF and word embedding vectors (100-item subset).

Sim.	Item Pair
TF-IDF — Most Similar	
1.000	*Resident Evil4 PS2 ↔ Resident Evil4*
0.587	*Halo2 ↔ Halo3*
TF-IDF — Least Similar	
0.000	*Metal Gear Solid ↔ Final Fantasy IX*
0.000	*Silent Hill ↔ Metroid Prime*
MiniLM — Most Similar	
1.000	*Resident Evil4 PS2 ↔ Resident Evil4*
0.787	*Rock Band ↔ Guitar Hero III*
MiniLM — Least Similar	
–0.096	*Legendary ↔ PS3 Wireless*
–0.084	*Final Fantasy ↔ PS3 Dualshock 3*

embeddings produce a more continuous scale because they can capture the semantic meaning between descriptions and not rely on exact word matches. Items can still be considered similar even if they use different but related words and very different words are pushed far apart. This is visible in the Table 5, where cosine similarity for TF-IDF drops to 0, whereas the embedding model produces negative similarities. This shows that word embeddings provide a finer grained notion of semantic similarity and can better separate unrelated items.

4.1 Week 10

In this week I develop a content-based (CB) recommender model. CB models move away from predicting purely based on other items or users, instead they use features, metadata, and qualities about an item to match and predict how a user will interact with it. It does this by converting the feature columns into a numerical vector representation of the item. So each item gets its own vector that encodes it in a vector space. User vectors are created by aggregating the vectors of the items the user has interacted with in the past. This is so that the user's representation reflects their preferences. To predict ratings I use cosine similarity between the user vector and an unobserved item. Working only with items that appear in the train and test set I used the columns description, features, categories from the metadata file. I chose these columns because they provide complementary information. Descriptions captures high level summary, categories provide genre and tags that group similar games, and features can capture finer details like platform or version. I omit title and detail columns because they are redundant and expect them to contribute marginal additional signal. All the columns are run through the same text preprocessing step as described in week 9 (lowercasing, removing non-alphanumeric characters, and stop-word removal). After, I convert each text feature into a TF-IDF vector and concatenate them for one final item representation. For user representations I use a weighted average of the item vectors so that items rated high by a user contribute more to their representation. This reflects the idea that items a user likes a lot says more about their taste. To tune the TF-IDF, I run a grid search on hyperparameters like minimum document frequency and maximum document frequency which control the vocabulary quality by removing overly rare or common tokens in every item. N-gram size which allows the model to include both single words and short phrases which can be more informative than individual words. Lastly, block weights that let me put more weight to some column group more than others. For each configuration in the grid, I build the TF-IDF item and users representations. Then I evaluate on the validation hold out set from the train set. I use MAP@10 as the metric to maximize on because it captures relevance and ranking. It rewards having relevant items in the top-k list and those items being high on the list. In practice I observed that when MAP@10 improves the other metrics increased as well. The goal was to prioritize relevance over minimizing error.

Param	Values Searched
`min_df`	1, 2, 3, 5, 7, 10
`max_df`	0.75, 0.85, 0.95, 0.99
`ngram_range`	(1, 1), (1, 2)

Model	HR	P@10	MAP@10	MRR	Cov.
CB	0.2268	0.0266	0.0222	0.0794	0.6947

Table 6: TF–IDF grid search hyperparameters and train set performance.

The best configuration achieved a value of 0.022 MAP with the following configuration: `min_df` = 5, `max_df` = 0.99, `ngram_range` = (1, 2), with block weights for description, features, category, and rating set to 1.5, 1.0, 0.5, and 0.1 respectively. Table 6 shows the full hyperparameter setup alongside the model's performance on the test set. The result show a hit rate of 22% and a coverage of 69%. Compared to TopPop, the CB model improves, Precision@10 by +0.003,

MAP@10 by +0.003, MRR by +0.009 and coverage by the largest margin +0.68. Hit rate stays about the same. Coverage improving means that CB includes a wider range of items in its recommendation. This suggests that by finetuning the TF-IDF hyperparameters and optimizing for ranking I was able to slightly improve above the popularity baseline. At the same time, the overall score remains low and unsuitable for commercial use. CB models also struggle with noisy metadata limiting how much they can improve over TopPop.

5 Week 11

In this week I implement two hybrid recommender models. Hybrid models are used to combine the best of both worlds of different models in hope of improving performance. Both hybrid models use the collaborative filtering SVD model and a CB model with a parallel combination strategy. I use the SVD model from week 7 as the base CF. I chose this because it performed better than KNN in all metrics except coverage. The first hybrid combines SVD with the CB recommender model from week 10 that uses the metadata columns, which I will refer to as regular CB. The second hybrid model uses a CB model that uses LLM generated descriptions instead of the original ones. The goal is to see if synthetic descriptions from titles and a short prompt can improve recommendation quality. For the LLM generated content I choose gemma3 and qwen. I wanted to compare gemma with 1.2 billion parameters and qwen with 4 billion to see if a complex model could produce more insightful descriptions.

To save computational cost I reduced the candidate item pool down from 927 to 599. I do this by taking, for each user, the top-20 SVD predictions, and then forming the union of all these items. This also means the hybrid can only recommend items from this reduced pool. The LLMs are then prompted to created a brief 1-2 sentence description about each video game based on the title. I set the temperature to 0 for predictable and deterministic output. The runtime was quite large. Both models took multiple hours to run so I saved the descriptions into a CSV file for reuse. After generation, I preprocessed the descriptions with the same method as week 9. I then create item and user TF-IDF vectors then make predictions on the unobserved items in the train set. For the combination strategy I test two parallel strategies, weighted sum (WS) combination and reciprocal rank fusion (RRF). The WS strategy takes the predicted scores from SVD and the scores from the CB model and combines them linearly to get a re-ranked combined score. RRF combines rank positions as opposed to using raw scores. It rewards items that appear high in both ranking regardless of score value. I included RRF to see how focusing on rank score compares to WS. I choose both methods to analyze the combination strategies. Table 7 shows the performance of all models with RRF and weighted sum evaluation. The SVD+regular CB model performs the best in all the metrics among all hybrid models. This is consistent with week 10, where the individual CB model had performed well improving over TopPop. In the WS hybrid I put more weight on the CB scores (0.7) and less on SVD scores (0.3). SVD+CB improves in all metrics over SVD, but remains close to CB showing how putting more emphasis on the CB score provides the best value in the hybrid model. The content-based signal dominates while SVD provides an extra boost for items that align with past user ratings. RRF for the same hybrid model shows a decrease in performance across all metrics. A likely reason is that neither SVD nor CB is very strong at ranking items and they are contributing noisy ranks which degrade performance.

Model	HR	P@10	MAP@10	MRR	Cov.
SVD + Regular CB	**0.2333**	**0.0272**	**0.0215**	**0.0762**	**0.6321**
SVD + Regular CB (RRF)	0.1670	0.0186	0.0163	0.0616	0.5933
SVD + qwen3 CB (WS)	<u>0.0734</u>	<u>0.0078</u>	0.0064	0.0246	0.5631
SVD + qwen3 CB (RRF)	0.0698	0.0075	0.0059	0.0231	0.5814
SVD + gemma3 CB (WS)	0.07	0.0073	<u>0.0081</u>	0.03	0.5641
SVD + gemma3 CB (RRF)	<u>0.0734</u>	<u>0.0078</u>	<u>0.0081</u>	<u>0.0298</u>	<u>0.5901</u>

Table 7: Evaluation metrics for SVD models combined with content-based components, comparing weighted-sum and RRF strategies. Best preforming LLM hybrid underlined.

For the LLM-based hybrids SVD+qwen3 and SVD+gemma3, the overall performance is much lower than regular CB hybrid. The scores are also below TopPop and regular CB model. The best LLM hybrid WS reveals qwen with has the highest hit rate (7.3%) and the higher precision (0.008) where as gemma WS is higher in MAP@10 (+0.0018) and MRR (+0.006). I had set the weights, 0.3 on SVD and 0.7 on CB. I kept more signal on the CB to assess the affect of LLM-generated features on the hybrid model performance. The findings show minimal varia-

tion in the LLMs used suggesting the description from the generations only have a marginal impact. When the two the RRF LLM hybrid models are compared head to head we see that gemma hybrid has a better hit rate, a 0.0003 increase in precision and a 0.002 increase in MAP@10. Within all LLM hybrids the gemma RRF had the highest MRR 0.029. and the highest coverage 0.59. When comparing the WS and RRF combination strategies, RRF slightly degrades performance for qwen hybrid compared to its WS counterpart. For gemma it yields only a very small improvement. This means that a simple weighted sum of SVD and CB scores better preserves the strong CB signal and modest rankings. A key limitation of the analysis is that the comparison between WS/RRF variants is bit shallow and not isolated from other factors. In future works I would focus on a single CB-based hybrid model and conducted more parameter tuning to isolate how the combination strategies affect performance. Overall the LLM hybrids are relatively similar in their performance. Having a larger model (qwen3) does not yield better recommender performance. I also observe that the choice of candidate pool size k plays a role. When I restrict each user to the top-10 SVD candidates, there is almost no room for re-ranking so the LLM hybrids add little value. Increasing k to 20 gave re-ranking more space to move items around, which helps improve MRR a bit, as seen with gemma+RRF model. Over all the performance of LLM hybrids do not beat TopPop recommender and generally perform low. A key reason is that generated descriptions were generic and not super insightful. Generation was done with a fix prompt and temperature set to 0, creating very deterministic template like text. Also the text preprocessing may have removed any remaining subtle differences. I did not tune the TF-IDF vector representation for the LLM text as I wanted to see these how descriptive the LLMs can be out of the box. In the end, the LLM generated metadata did not improve recommendation performance over hybrid + baseline CB or popularity baseline. To improve this, I would have designed prompts to include certain fields like genre, feature, categories. I would also fine-tune the TF-IDF hyperparameter and consider switching to embedding based representation.

Model	HR	P@10	MAP@10	MRR	Cov.
TopPop	0.2133	0.0236	0.0190	0.0709	0.0107
KNN	0.016	0.0016	0.0009	0.0050	**0.8900**
SVD	0.0553	0.0057	0.0051	0.0180	0.4757
CB TF-IDF	0.2268	0.0266	**0.0222**	**0.0794**	0.6947
SVD + CB	**0.2333**	**0.0272**	0.0215	0.0762	0.6321
SVD + CB (RRF)	0.1670	0.0186	0.0163	0.0616	0.5933
SVD + qwen3:4b CB	0.0742	0.0080	0.0064	0.0249	0.5631
SVD + qwen3:4b CB (RRF)	0.0698	0.0075	0.0059	0.0231	0.5814
SVD + gemma3 CB	0.0706	0.0074	0.0077	0.0294	0.5707
SVD + gemma3 CB (RRF)	0.0734	0.0078	0.0081	0.0298	0.5901

Table 8: Full comparison of all models ($k = 10$) and relevance ≥ 3, grouped by approach. Best per column in **bold**.

6 Final Comparison

Table 8 shows the final comparison of all the models. A clear pattern is that the CB regular model is the strongest model. Individually it performs better than TopPop and has the highest MAP@10 and MRR. The SVD + CB hybrid WS achieves the highest hit rate and precision among over the CB individually, with MAP@10 and MRR very close. This indicates that combining with SVD gives a small ranking boost while keeping coverage reasonable. Most of the gains still come from CB as seen in earlier weeks where SVD struggled with sparsity and CB is set to the higher WS factor. The CB model is designed to mitigate these sparsity issues by leveraging textual metadata. KNN again dominates coverage but this good performance is deceptive as all its other metrics perform the worst showing case its flaws when dealing with long-tail and sparsity analysis. The LLM-based models perform better than the CF models but could not meaningfully provide better recommendations than the TopPop or CB baseline. TopPop remains surprising strong baseline in terms of ranking but almost no coverage, showcasing the popularity bias. The CB regular and SVD + CB hybrids offer the best trade-off between accuracy and coverage, but still rely on simple features. Overall none of these models reach a level suitable for a commercial system. Given the observations from week 9, using more modern sentence embeddings in the text would likely allow for better granular signals and cosine similarity which would improve all the CB and hybrid models in future work.

In the end many fine-tuning experiments can be conducted, but the key is understanding the trade-offs between accuracy and coverage and quality of the dataset.

References

Maria Maistro and Christina Lioma. 2026. Web recommender systems: Lecture slides, university of copenhagen.

Wenhui Wang and 1 others. 2020. all-MiniLM-L6-v2: Sentence transformer model. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2. Accessed: 2026.