Web Recommender Systems Project, DIKU, Block 3, 2025

Kasper Nicolaj Schiller

Abstract

In this paper, we implement and evaluate several recommendation systems based on a musical instrument dataset from Amazon. Our models underperform compared to a frequency-based benchmark. In the end, we discuss the models' limitations and propose directions for future work.

1 Data

We load the smaller version of the "Musical Instruments" dataset from the Amazon Review data 2023 (5-core subset), which contains two splits: one for training, and one for testing. Each split contains four columns. The *item_id* column is a unique 10-char identifier for products, and *user_id* is a unique 28-char identifier for reviewers. The *rating* column is an integer score (1-5) assigned by the reviewer, and the time of review can be found in the *timestamp* column in a unix format.

We clean both splits, and conclude that there are no missing or misformatted values in either of the columns. We also check for duplicates in both splits; if a user have rated an item more than once, we keep the last occurrence. We also double-check that every user from the test split appears in the training split: this is indeed the case.

A summary of user and item statistics are shown in Table 1. The statistics suggest that high ratings are dominant in the dataset. This may impact recommendation performance negatively, as it can be hard to account for absolute differences in user ratings. One way to get around this is to avoid evaluation metrics like RMSE or MAE and focus on utility-based methods.

The dataset is relatively sparse, as it contains only 9913 reviews for 800 users and 509 items - this may impact model performance.

In addition, the dataset has quite a high imbalance: 22% of the items account for 50% of ratings, which may lead to popular items dominating recommendations. 35% of users also account for 50% of the reviews: they too may have too much impact on the models predictions. As seen in Figure 1, the dataset is highly exposed to the impact of the long tail.

Metric	Number
Reviews	9913
Users	800
Items	509
Avg. reviews/user	12.39
Items for 50% of reviews	111 (22%)
Users for 50% of reviews	279 (35%)
Sparsity	97.6%
Rating distribution	$\mathcal{N}(4.54, 0.69)$

Table 1: Summary of the cleaned train split.

In Table 2, we show the frequency of items rated higher than 2. We will use this recommender system as benchmark against other models developed in this paper.

Item ID	Frequency	Average Rating
B0BSGM6CQ9	153	4.67
B0BPJ4Q6FJ	153	4.74
B09857JRP2	141	4.77
B0BCK6L7S5	120	4.30
B0BTC9YJ2W	119	4.74

Table 2: The top 5 items sorted by frequency of ratings above 2.

2 Collaborative Filtering

In this section, we implement two collaborative filtering recommender systems with the purpose of predicting unobserved ratings in the training set. They are applied to rank non-rated items for each user in the training set.

A KNN Baseline model. We employ the KNN Baseline model from the *surprise* library. The

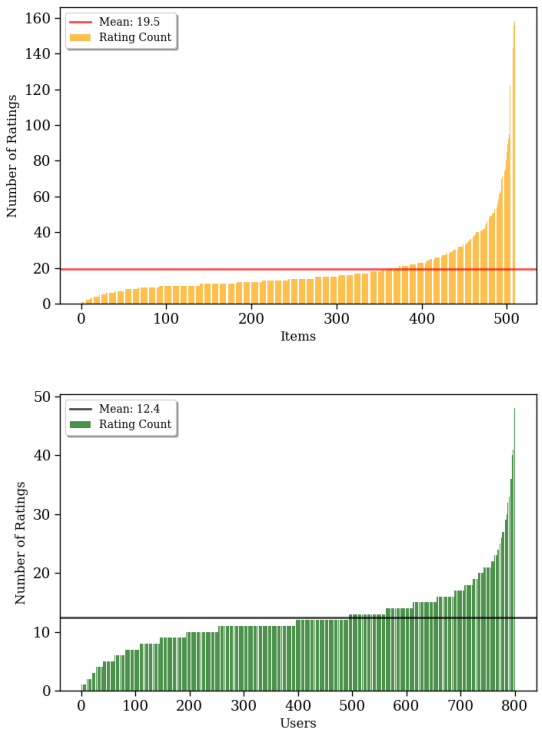

Figure 1: Number of reviews over items and users.

model takes into account a baseline rating consisting of the deviations of both the item and user from the average rating (Koren, 2010). For our purpose, we use the alternating least squares algorithm for computing this baseline with 10 epochs and a regularization parameter of 15 and 10 for the users & items, respectively. The search space for fine-tuning is shown in Table 3.

Table 3: Search space table for KNN Baseline.

Parameter	Tested Values & Options
k (Neighbors)	3, 5, 8, 10, 15, 20
Approach	User-based, Item-based
Similarity Measures	Cosine, MSD, Pearson, Pearson Baseline

A SVD model. Our latent factor model of choice is the SVD. As our dataset is sparse, we initialize the model with a higher regularization parameter of 0.1 to prevent overfitting. The search space for the model is shown in Table 4.

Table 4: Search Space Table for SVD.

Parameter	Tested Values & Options
Latent Factors	5, 10, 20, 30, 40
Epochs	10, 20, 30, 40, 50, 60

We choose to fine-tune the models with the target of predicting unobserved ratings. This decision has several drawbacks, as we are to rank items based on these models. Thus, for future work, a ranking-measure such as nDCG should be considered for fine-tuning.

Both RMSE and MAE are valid options to optimize our models for predicting unobserved ratings. MAE penalizes wrong recommendations linearly, where the value of RMSE can be impacted by outliers. Both are limited by the impact of the long tail, which is definitely a factor in our sparse data set. One way to solve this problem is to compute the RMSE and MAE separately for each item and compute the weighted average. For now, we choose the MAE to increase overall accuracy, computing the average over 5 folds. We do this on the training set using the *surprise.model_selection* library, to tune hyperparameters in the search spaces.

The results for the KNN Baseline Model are shown in Table 5. The lowest averaged MAE over the 5 folds was accomplished using the item-based approach, having $k = 10$ using mean squared difference for similarity. With a MAE of 0.569, the predictions are off by more than half a rating on average. The optimal parameters for the SVD model was 20 latent factors and 50 epochs, as shown in Table 6.

With the optimal hyperparameters, the models were fitted on the full training set. Then, for each pair of users and items not present in the training set, we perform predictions on their ratings. The ratings are then sorted in descending order to obtain their rankings.

3 Evaluation

The purpose of this section is to discuss evaluation measures for recommender systems, and to evaluate the CF models presented with their optimal hyperparameters found in Section 2. Evaluation of all models proposed in this paper is shown in Table 7.

The RMSE Metric. First, we consider the RMSE metric to evaluate the CF models. For each pair of users and items in the test set, we predict ratings based on our two respective models, and evaluate performance using the metric. We obtain an RMSE of 1.081 for the KNN Baseline model, and for the SVD model, the value is 0.992.

At first hand, the models' performance may seem rather problematic. Assume that the test set has

Table 5: MAE averaged over 5 folds for KNN Baseline.

	User-Based				Item-Based			
	Cosine	MSD	Pearson	Pearson BL	Cosine	**MSD**	Pearson	Pearson BL
k=3	0.629	0.615	0.626	0.616	0.589	0.584	0.604	0.597
k=5	0.622	0.608	0.621	0.616	0.581	0.582	0.597	0.593
k=8	0.616	0.605	0.619	0.609	0.577	0.578	0.600	0.597
k=10	0.613	0.601	0.621	0.620	0.572	**0.569**	0.596	0.595
k=15	0.614	0.603	0.619	0.611	0.576	0.581	0.605	0.602
k=20	0.610	0.604	0.619	0.618	0.574	0.574	0.600	0.605

Table 6: MAE averaged over 5 folds for SVD.

	Latent Factors				
Epochs	5	10	**20**	30	40
10	0.775	0.776	0.775	0.776	0.776
20	0.761	0.761	0.761	0.763	0.762
30	0.755	0.754	0.756	0.756	0.757
40	0.753	0.754	0.753	0.753	0.754
50	0.753	0.753	**0.751**	0.752	0.752
60	0.752	0.753	0.752	0.753	0.751

a variance of 0.69, just like the training set (see Table 1). Then, the RMSE suggests that our models perform worse than predicting the global average. However, this metric has limitations. Firstly, the metric is heavily impacted by ratings on popular items due to the high sparsity of other items, also known as the impact of the long tail (Aggarwal, 2016, p. 241). In addition, the metric does not take into account the ranks of the recommendations.

Rank-based utility measures. Let us now consider rank-based utility measures. They have an advantage when recommending items to users as they tend to be more realistic. To initialize, we create a *ground truth vector* for each user, where ratings ≥ 3 for an user-item pair in the test split are considered relevant.

The mean *Hit Rate* is the proportion of users that was recommended an item relevant to them on average. The metric does not consider the order of recommendations, which is a drawback in various domains. *Precision at k*, which we will denote $\boldsymbol{P@k(u)}$, reveals the proportion of how many relevant items were present in the top-k recommendations for the user u. With a value of 1, all top-k items were relevant, and with a value of 0, no items were relevant. This metric has a major drawback when the order of the top-k ranks matter for the evaluation, as the order is not considered either in the computation. *The Reciprocal Rank* $\boldsymbol{RR@k(u)}$ metric is the inverse of the rank of the first relevant item in the top-k recommendations for user u. Therefore, in contrast to the two previous metrics, this measure takes into consideration the order of the top k ranks. A limitation is that it puts no weight on the relevance of items beyond the first recommended relevant item.

Summing the $P@k(u)$ metric over the relevant recommended items, where k is their respective rank positions, and dividing the result by the number of relevant items yields the *Average Precision* metric, denoted $\boldsymbol{AP@k(u)}$. This metric solves one limitation: it considers the order of the top-k relevant items, which is crucial in most domains on the web.

A drawback of all of these utility-based metrics is that they are based on a ground-truth vector which does not consider relative relevance.

Finally, the *Coverage* measures the proportion of items that were recommended by the system, out of all items that could have been recommended. This metric may reveal a problem that the utility-based methods do not: the impact the long tail has on the recommendations.

We use the metrics on the rankings created in Section 2 to evaluate the CF systems as shown in 7. The cutoff is $k = 10$, thus we only include the top-10 recommendations for each user.

For both CF models, 9.2% of the users had a relevant item recommended in their respective top-10 recommendations, per the hit rate. For TopPop, this number is 25.4%. The low P@10 for all of the models may be due to the fact that they do not consider cases where a user has less than k relevant items. If a test user has only one relevant item in the test set, the $P@10(u)$ metric for that user will either conclude to 0.1 or 0.0. In addition, for the CF models, the top-k recommendations may also have the same predicted rating, making the order of those recommendations arbitrary. This is a drawback of training with MAE or RMSE metrics.

It is clear that the TopPop recommender system is outperforming our models according to all met-

Table 7: The mean of a metric was computed across all users. All items are used to compute the coverage.

	Mean HR@10	Mean P@10	MAP@10	MRR@10	Coverage
TopPop	0.254	0.032	0.034	0.116	1.93%
KNN Baseline	0.092	0.010	0.010	0.035	63.9%
SVD	0.092	0.010	0.009	0.027	28.4%
Word2Vec Content-Based	0.096	0.011	0.008	0.035	40.0%
Parallel Hybrid	0.104	0.011	0.007	0.024	57.7%
Llama3.2 Content-Based	0.106	0.011	0.011	0.037	40.3%

rics apart from the coverage. One reason for this is that predicting the most popular items is not a bad estimate. Also, the sparsity of our dataset is high, making it hard to train models to do the task.

A limitation of the TopPop model is that it lacks personalization. It has a low coverage as it recommends the same 10 items every time, which may be a problem given a new dataset. The SVD achieves a higher coverage, but still rather low at 28.4%. This stands in contrast to the KNN Baseline model, which has a much larger coverage, 63.9% partly due to its b_{ui} parameter, which takes in consideration both the average rating over the user and the item. This is a common aspect of neighborhood models: they handle users incrementally, making them better at recommending new items on sparse data.

Finally, a drawback of a latent factor model like SVD is that it is not as explainable as neighborhood models. In addition, a transductive model like SVD may require retraining or approximation methods (like averaging over users) to apply it to new users/items. This makes them more computationally expensive. Neighborhood CF models often do not have this requirement, as they can handle new users or items incrementally without full retraining: KNN does this by computing the mean squared difference to existing items when a rating of an item is to be predicted. Yet, latent factor models are often more accurate because while neighborhood models capture local patterns, latent factor models are often able to find global patterns in the data.

The impact of the long tail. To evaluate the impact of the long tail, we compute the coverage & hit rate for respectively the top and bottom 20% of users, sorted by number of ratings in the training set. The result is shown in Table 8.

We see that the TopPop model achieves a high hit rate for the bottom 20% users. This shows that users that have bought fewer items tends to have bought a high proportion of popular ones.

Table 8: The hit rate and coverage for the top 20% users and items, respectively.

	HR@10		Coverage	
	Top	Bottom	Top	Bottom
TopPop	0.050	0.350	9.80%	0%
KNN	0.025	0.080	46.1%	96.1%
SVD	0.031	0.130	45.1%	28.4%

Meanwhile, the users that have bought more items may have more often bought items that are niche.

The CF models have higher hit rate on the bottom 20% of users which stands in contrast to the fact that it should be easier to recommend items to users that have more relevant items, since we have more data on those users. This may be because the high sparsity for those users make the models recommend items that are popular across all users, which is a good estimate according the results from TopPop.

SVD achieves a higher coverage for the top than the bottom. This indicates that the SVD model often recommends more popular items. For the bottom 20% items, the KNN model achieves the highest coverage by a high margin. The result harmonizes with the fact that our KNN model is a neighborhood model and should therefore more often recommend long-tail items.

Error Analysis. For the KNN Baseline model, we now analyse the characteristics of users that our model performs strongly on, and those where it underperforms. To do this, we compute the Reciprocal Rank for every user in the training set with $k = \infty$. We choose two reference users: *AFWEOWK32WTIBX4MC7D3WFDLCKJQ* with an RR of 1.0, and the user *AHL4BCH2O33HFN7KSBTIJUHCJYDA* with a low RR (0.04). Their characteristics are shown in Table 9.

We see that the high RR user has a neighbor overlap of 54.5%. This may be because that user simply has rated way more items (33) than the

Table 9: Characteristics of reference users. The ten nearest neighbors of the users were found using the cosine similarity. An overlap occours for an item if it has been rated by at least one neighbor.

	High RR user	Low RR user
Items rated	33	6
Rating distribution	$\mathcal{N}(4.48, 0.74)$	$\mathcal{N}(5.0, 0.0)$
10-Neighbor overlaps	18 (54.5%)	6 (100%)

global average (12.39, as per Table 1). The low RR user has a 100% neighbor overlap, which indicates that the user may only have rated popular items.

The user with a low RR has rated all items 5. This may be the problem for the predictions: our KNN is using the item-based approach, but many users have rated every single item the same. Let us further investigate this hypothesis by computing the rating distribution across groups.

First, we divide the users into 3 groups: Low, Medium and High RR as seen in Table 10. Based on the general distribution of low-RR users, it is clear that our low-RR reference user in an outlier in terms of rating distribution. In addition, neither the rating distribution or number of items rated is correlated with the Reciprocal Rank. For further analysis we propose the **MA@k** metric, defined as the *mean arbitrarity* by Equation (1),

$$MA@k = \frac{1}{N}\sum_{i=1}^{N} A@k_u \tag{1}$$

$$A@k_u = \mathbb{I}\left(\hat{r}_{u,1} = \hat{r}_{u,k}\right) \tag{2}$$

where N is the amount of users, and $\hat{r}_{u,j}$ is the predicted rating for user u for the item at rank j.

Since the top-k rankings are ordered descendingly by their ratings, the indicator function defined in Equation (2) returns 1 if of all the top-k items for a user have the same associated rating prediction. This definition arises from the fact that the order of the top-k rankings are considered arbitrary if their predicted ratings are equal.

With an MA@20 of 0.68 for the low RR group, 68% of these users' top 20 rankings are arbitrarily ordered. The values show a negative correlation between the Reciprocal Rank and MA@20. This suggests that these users may have a low RR because the predicted ratings for the top-20 items are often the same, thus their order is arbitrary. The nDCG measure being a ranking metric could have solved this problem had it been used during training instead of the MSE. However, the KNN model may still have ranked the top-20 items arbitrarily due to sparsity.

Table 10: Characteristics of users in RR groups. $Low \leq \frac{1}{20}$, $Mid \in \{x \mid \frac{1}{20} < x \leq \frac{1}{3}\}$ and $High \in \{\frac{1}{2}, 1\}$.

User Group	Low RR	Mid RR	High RR
Amount of Users	378	52	7
Rating Distribution	$\mathcal{N}(4.50, 0.72)$	$\mathcal{N}(4.59, 0.65)$	$\mathcal{N}(4.37, 0.84)$
Avg. Review Count	10.7	11.3	10.6
MA@20	0.68	0.7	0.29

4 Text Representation

We now investigate two NLP techniques to represent items in a vector space and explore their similarity based on text characteristics: TF-IDF and Word2Vec. We load the full *metadata* dataset from Absalon, which includes **23984** musical products, also from the Amazon Review data. It associates each item with two features: title and a description. In order to have full context for the TF-IDF model, we do not filter out items.

Initially, we preprocess the *title* and *description* feature for each item as follows: lowercasing $\rightarrow$ tokenizing $\rightarrow$ stopword removal. We define the stopwords to be the union of 1) the english NLTK stopword corpus and 2) the punctation list from the string library. For tokenizing, we use the NLTK library as well. As the TF-IDF measure estimates the importance of each word in our context, we add a last step for preprocessing the features for this model: *stemming*, using the PorterStemmer algorithm from the NLTK library. Stemming reduces similar words to the same written form (Lovins, 1968), which makes it useful for the TF-IDF measure. Lemmatization, which groups similar words to one meaning, could also have been used, but stemming is cheaper computationally. Word2Vec has pre-trained embeddings, thus we will not be either stemming or lemmatizing the description column to prepare for this embedding. In Table 11, we show attributes of the metadata dataset, before and after preprocessing.

Table 11: Summary of the metadata with the 23984 items. The *Cleaned Vocabulary* reflects the vocabulary size after lowercasing all words and removing stopwords. The *Stemmed Vocabulary* is the size after applying stemming to the cleaned data.

	Title	**Description**
Original Vocabulary	33193	74053
Cleaned Vocabulary	29987	61332
Stemmed Vocabulary	27962	52688

We compute the TF-IDF measure for each item using the implemetation from *sklearn.feature_extraction.text*. This results in a 23984×52688 sparse matrix, which corresponds to the number of items $\times$ stemmed vocabulary size, as expected.

For the Word2Vec embedding, we represent items using the pre-trained vectors *word2vec-google-news-300* from the *gensim* library, which contains 300-dimensional word embeddings for 3 million words, obtained using the skip-gram model architecture proposed in (Mikolov et al., 2013). For each item, we average the embedding on the column dimension. The similarity between items is obtained by computing their cosine similarity in the TF-IDF matrix and in the Word2Vec embeddings, respectively. Both computations result a 23984×23984 matrix, as expected.

Let us now investigate the models' output by example. The stemmed description of *B0002D01KO* is *['gigbag', 'epiphon', 'solidbodi', 'electr']*, while the description of *B0002D0CCQ* is *['gibson', 'string', 're-inv', 'modern', 'guitar', ...]*. They share no common words after stemming, resulting a zero similarity according to the TF-IDF measure. However, as Word2Vec captures semantic relationships between words without stemming, we obtain a similarity of 0.98 for this representation, which may be because *guitar*, *epiphon*, *gigbag* are considered similar: the latter two are guitar brands. This shows a limitation of the TF-IDF for tasks that require semantic understanding of words outside the context. However, TF-IDF may still outperform Word2Vec as it takes into consideration the context of the words.

5 A Content-Based Recommender System

In this section, we build a content-based recommender system based on the Word2Vec embeddings computed in the previous section. This embedding was chosen as it contains pre-trained word embeddings to which enables capturing semantic relationships between words. However, the choice of TD-IDF is just as justifiable. We filter the data to only include the 509 items from the training split. We use the *title* and *description* of each item as features for the model. We use the preprocessing in Section 4, prior to stemming.

Using only the training set, we represent each user by the computing average Word2Vec embedding of the items they have interacted with, weighted by the rating they gave those items. Thus, each user has two features, which is a vector of size 300 each. This represents the users profile.

The similarity between user u and item i is computed using the cosine similarity by

$$sim(u,i) = \alpha \cdot cos(v_u^t, v_i^t) + (1-\alpha) \cdot cos(v_u^d, v_i^d) \quad (3)$$

where $\alpha = \frac{2}{3}$. $v_j^t \in \mathbb{R}^{300}$ and $v_j^d \in \mathbb{R}^{300}$ corresponds to the embedded title and description vectors, respectively, for the user or item j. This means that similarity on the *title* feature is weighted higher than the *description* feature. The choice of α was made by an assumption that the *description* feature contains noise, and thus increases the similarity between items that are not related. Cosine similarity was chosen because Word2Vec captures the semantic meaning of the words through the direction of the vector, making magnitude redundant for our task.

For each user, the non-rated items are sorted descendingly according to Equation (3) to retrieve rankings. After, the similarity values are normalized to the original rating range using min-max normalization to prepare for a hybrid recommender system.

Evaluation. We evaluate the systems with the metrics presented in Section 3. The results are shown in Table 7. The content-based system performs worse than the collaborative filtering systems according to the MAP, but achieves better results according to the other metrics. One reason could be the similarity function proposed in Equation (3). We should consider either fine-tuning the α-parameter or concatenating the features. Leveraging the large amount of context we have in the metadata is also an option by turning to the TF-IDF measure. Finally, the model is also limited by the high sparsity of the data set.

6 Hybrid Recommender Systems

In this section, we build two models.

A Parallel combination strategy. The first model is a hybrid and combines the outputs of the KNN Baseline model and the content-based model presented in Section 5. The recommendation rankings for each user is the predicted ratings ordered descendingly according to the weighted sum, with $\alpha = \frac{1}{3}$ (KNN) and $\beta = \frac{2}{3}$ (Content-Based).

A LLM-Content-Based Model. The second model is based on Word2Vec embeddings of LLM-

generated item descriptions. We use the Llama-3.2 from (Meta, 2024) with 1.24 billion parameters.[1] We chose this model because it outperforms some larger models despite being lightweight, and as (Strubell et al., 2019) suggests, lightweight models should be prioritized in NLP research to lower carbon emissions. The model was initialized using the Pipeline API from Huggingface, using the parameters shown in in Table 12.

Table 12: Parameters for the Llama-3.2-1.B model.

Parameter	Value
Initial prompt	*"text-generation"*
Temperature	0.7
Nucleus probability threshold	0.9
Maximum new tokens	50
Repetition penalty	1.2
Sequences	1

For each item present in the training set, we execute the following prompt dynamically, with each item's title as input:

Generate a detailed and accurate description for the following musical instrument: {item title}.

Now we have no empty descriptions. This was not the case in the original data. In addition, the descriptions are longer. We find that they are much like the original descriptions in the content, but not in the design. For example, for item *B007MY5BDI*, the LLM-generated description is just a review: *"The strings on this set are an improvement over any other I've used before..."*, which seems like a review. The original descriptions are more of a marketing text, possibly from the seller: *"80/20 Bronze are our brightest acoustic strings, made to give your guitar an unparalleled shine..."*.

We turn to Word2Vec embeddings to represent the items in a vector space. We use exactly the same setup as in Section 4 and 5 in regards to preprocessing and ranking, apart from now only using 1 feature: the LLM-generated descriptions.

Evaluation. The results for the models are shown in Table 7. Clearly, the hybrid systems are also limited by the sparsity of the data set. In addition, we see that the model with LLM-based descriptions outperforms every other model according to the average precision, apart from TopPop, including the models using the original description. This is even though the parallel hybrid model has an extra feature and also considers the KNN outputs. The reason may be that the LLM generated descriptions for all products, resulting in no cases of empty features. We should consider other aggregation methods for the parallel hybrid, such as Reciprocal Rank Fusion, or fine-tuning the α and β parameters for the weighted sum.

[1] Running this model requires an access token. Granted from (Meta, 2024).

7 Discussion

The TopPop model achieves the highest scores for all metrics apart from coverage. It is clear from the summary in Table 1 that sparsity is an issue. Also, the dataset is impacted by impact of the long tail, as seen in Figure 1.

A high sparsity has a negative impact in particular for neighborhood-based and matrix factorization models, as they rely on user-item interactions. To improve the CF models' performance, we suggest training them on the nDCG metric proposed in (Järvelin and Kekäläinen, 2002) instead of the MAE. We saw in Table 10 that training the models to predict ratings may be the reason why some users have a low reciprocal rank. The two CF models can also be combined as in (Koren, 2008).

For the content-based model, we should consider filtering out the items that hold invalid features, such as an empty description. In addition, the TF-IDF measure from Section 4 should be considered, as the metadata equip us with context from 23984 items, which Word2Vec does not utilize. The Parallel Hybrid model does not perform better than the models it combines. Two solutions to this are proposed in Section 6. For increased performance, graph-based recommender systems should also be explored, such as Personalised PageRank.

Finally, to further evaluate performance, the nDCG metric should be computed for each system.

8 References

References

Charu C. Aggarwal. 2016. *Recommender Systems*. Springer International Publishing. ISBN 978-3-319-29657-9.

Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):422–446.

Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In

Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 426–434. ACM.

Yehuda Koren. 2010. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data*, 4(1):1–24.

Julie Beth Lovins. 1968. Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, 11:22–31.

Meta. 2024. Llama-3.2-1b. Accessed: 2024-03-26.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *arXiv preprint*.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy.