Report for Recsys Challenge 2024

GRANTNER TOBIAS, KLEINEGGER MAXIMILIAN, MAHLER LUKAS, MURIASOVA KARINA, and TAHA JOSEF, TU Wien, Austria

This year's group project consists of taking part in the *ACM RecSys 2024 Challenge*. The challenge's main aim is to predict which articles a user will click on, based on their click history, user or session metadata, and the articles' content. We are dealing with the Ekstra Bladet News Rcommendation Dataset (EB-NeRD), a Danish dataset created by Ekstra Bladet for news recommendation research. The task included implementing three models: a baseline provided by the challenge team, an algorithm assigned by the course team, and an algorithm we could choose freely. After implementing and hyperparameter tuning all algorithms, the assigned algorithm Deep Knowledge-Aware Network (DKN) turned out to outperform the Neural News Recommendation with Multi-Head Self-Attention (NRMS) (baseline) and the Long- and Short-term User Representations (LSTUR), our own chosen one. After obtaining initial AUC results of 0.6196 for DKN, 0.5472 for NRMS, and 0.5562 for LSTUR, we decided to fine-tune only the best-performing model. By providing suitable Danish Word2Vec embeddings instead of widely available and unsuitable English ones, we improved our model's AUC to 0.6956. Therefore, having a model that can deal with news recommendations.

Contents

Authors' Contact Information: Grantner Tobias; Kleinegger Maximilian; Mahler Lukas; Muriasova Karina; Taha Josef, TU Wien, Vienna, Austria.

1 Introduction

This year's group project consists of taking part in the *ACM RecSys 2024 Challenge*[1]. It focuses on online news recommendations. Therefore we have to deal with modeling user preferences based on implicit behavior, accounting for the influence of the news agenda on user interests, and managing the rapid decay of news items. To be more precise, the challenge aims to predict which articles a user will click on, based on their click history, metadata of the user or session, and the content of the articles itself.

The main tasks of this group project are: First, taking part as a group (**leaderboard name: GKMMT**) within the *ACM RecSys 2024 Challenge* , by uploading resulting predictions as a submission. Second, we had to implement three different algorithms:

- an assigned baseline algorithm,
- an algorithm assigned by the course team,
- and an algorithm from which we can freely choose.

Lastly, these algorithms should then be optimized and further tuned to be able to make good predictions. Therefore, this report gives insights into the approach, the results, and the difficulties we have encountered while implementing those three algorithms and taking part in the challenge.

2 Data

The dataset used for the RecSys challenge is Ekstra Bladet News Rcommendation Dataset (EB-NeRD)[2], which is a Danish dataset created by Ekstra Bladet for news recommendation research.

Eb-NeRD dataset includes active user interactions from active user logs with Danish news articles over a 6-week period, from April 27 to June 8, 2023. It contains features such as title, abstract, body, categories, and more. Active users were defined as individuals who clicked on news articles between 5 and 1,000 times during a three-week period from May 18 to June 8, 2023. Furthermore, EB-NeRD contains comprehensive information about user interactions with news articles including Impression ID, User ID, Article ID, Session ID, Inview Article IDs, Clicked Article IDs, Time, Readtime, Scroll Percentage, Device Type, SSO Status, Subscription Status, Gender, Postcode, Age, Next Readtime, Next Scroll Percentage in the `behaviors.parquet` file, and user click histories (User ID, Article IDs, Timestamps, Read times, Scroll Percentages) in the `history.parquet` file.

3 Approach

This section describes the methods we used, the reasoning behind them, how we compared them, and what measures we took to improve the used models for the *ACM RecSys 2024 Challenge*. For more detailed information on the implementation itself have a look at the documented source code[3].

3.1 Neural News Recommendation with Multi-Head Self-Attention (NRMS)

This section describes the algorithm we should use as a baseline for the *ACM RecSys 2024 Challenge* in order to have a model to compare the assigned and selected algorithms against. The organizers from the *ACM RecSys 2024 Challenge* kindly provide a repository with an implementation of it[4]. The main challenge here is to get it up and running and to

[1]https://recsys.eb.dk/
[2]https://recsys.eb.dk/dataset/
[3]https://gitlab.tuwien.ac.at/recsys-laboratory/teaching/24ss-recsys-lecture/Group_33
[4]can be found here: https://github.com/ebanalyse/ebnerd-benchmark/tree/main

make some predictions. This should normally be a rather easy task, but this sounds too good to be true, so we had to have some troubles, which we will discuss. So, what recommendation model should be used as the baseline algorithm? The ebnerd-benchmarking team provided an implementation of the Neural News Recommendation with Multi-Head Self-Attention (NRMS), which we will use as our baseline. This model should be generally well-suited because it is a content-based approach and therefore should be able to deliver some good rankings. It aligns the recommendation process with users' existing interests and preferences by identifying and suggesting articles with similar content, enhancing user engagement

Neural News Recommendation with Multi-Head Self-Attention (NRMS) [6] (as seen in figure **??**) is a recommender system that uses a news- and user-encoder. The news encoder uses multi-head self-attention to learn news representations from news titles by modeling the interactions between words and the user encoder learns representations of users from their browsed news. Furthermore, it uses multihead self-attention to capture the relatedness between the news. Lastly, additive attention improves news and user representations by selecting more important words and news.

3.1.1 Implementation. The implementation consisted of two parts: First trying to understand the given code and second getting it up and running. The first part already took some time, but after working with the data and understanding the algorithm wasn't that hard anymore. The bigger challenges arose with setting up an environment that can be easily set up, is easy to use, should allow reproducibility, and needs to work on a system with limited user permissions. We found a good working solution with poetry[5] and afterward had to make some small adjustments, which can be seen in the source code to make it run.

The big advantage of the ebnerd-benchmarking implementation, besides having a working model, is the provided dataloader, which is highly optimized and saves a lot of work that would otherwise need to be done. Nevertheless, we should examine the data we need to provide to the NRMS model.

- User History: The articles the user has clicked in the past.
- User Behaviors: The sessions per user, including the *inviewed articles* and *clicked articles*
- Articles: The articles include features like title, body,...

These three data sources are now preprocessed and combined with the Dataloader. First, we generate Word2Vec-embeddings of the title and sub-title column. Here, you could simply take some pre-trained GloVe embeddings[4], but those most likely lack effectiveness, because most of them concern the English language and we deal with the Danish language. Therefore, we used Huggingface and a pre-trained model to create those. Afterward, we include the user history into the user behaviors and generate labels based on the *Inviewed articles* and *clicked articles*. NRMS now needs two more things: First, we need a mapping between user IDs and unique IDs and secondly, we need a mapping between the words used in the embeddings and unique IDs. This can be easily seen in the original implementation from the recommenders-github[6]. The ebnerd-benchmarking implementation does this automatically for us and we don't need to bother with it. Lastly, they try to boost the performance by replacing the article IDs directly with the corresponding Word2Vec embeddings within the Dataloader. Therefore avoid providing it separately and doing it more cleanly. Now with all the necessary data provided and basic hyperparameters set, as seen in the next section, training can start!

[5]https://python-poetry.org/
[6]https://github.com/recommenders-team/recommenders/

3.1.2 Optimizations. We need a way to evaluate how well NRMS is suited for our task. Therefore, we performed some hyper-parameter tuning to assess the potential of this algorithm. We used the following configuration (see table **??**) and tested it with the training- and validation-set of ebnerd-small:

Parameter	**Description**	**Values**
history_size	Size of history considered	10, 50, 100
n_users	Number of users considered	20k, 50k, 70k
title_size	Max title length that considered	10, 50, 100
learning_rate	Learning rate of NRMS	0.0001, 0.001, 0.01
dropout	Dropout rate of NRMS	0.1, 0.3, 0.5

Table 1. Parameter Grid for NRMS

We got very similar results for all configurations varying only by around 4%. The best hyper-parameters were $history\_size = 100$, $n\_users = 70,000$, $title\_size = 50$, $learning\_rate = 0.01$ and $dropout = 0.5$, with an AUC of 0.556. We stopped further optimizations because our assigned algorithm DKN already outperformes NRMS at this point and we wanted to focus our optimization efforts on DKN.

3.2 Long- and Short-term User Representations (LSTUR)

This section describes the algorithm, we could choose freely to compete in the *ACM RecSys 2024 Challenge*. Our general approach was to find an efficient, but effective way to rank the *Inviewed Article IDs* per session. For this reason we

LightGBM First, looked at LightGBM, which relies on highly efficient gradient boosting decision tree [3]. However, we quickly realized the limitation of this approach, because processing word- and entity-embeddings with decision trees is not the best approach and does not work very well. Therefore, we dismissed it, because we want to integrate the articles content to make good predictions.

GRU Secondly, we looked at GRU [2], because it integrates short and long term user interests. However, it is a *Collaborative Filtering*-approach, which is not the best fit for this task. Because we have rather sparse metadata, where we can look for similarity and all the information lies in the content of the article itself. Therefore, adding content embeddings would have required too many changes to the model, which we couldn't do within the given time frame and available hours.

LSTUR After some more research, we ended up using the Long- and Short-term User Representations (LSTUR)-model [1].

LSTUR as seen in figure **??** "is a news recommendation approach capturing users' both long-term preferences and short-term interests. The core of LSTUR is a news encoder and a user encoder. In the news encoder, we learn representations of news from their titles. In user encoder, we propose to learn long-term user representations from the embeddings of their IDs. In addition, we propose to learn short-term user representations from their recently browsed news via GRU network. Besides, we propose two methods to combine long-term and short-term user representations. The first one is using the long-term user representation to initialize the hidden state of the GRU network in short-term user representation. The second one is concatenating both long- and short-term user representations as a unified user vector." [1]

Therefore to shortly summarize, LSTUR captures both short- and long-term preferences, by using embeddings of users' IDs to learn long-term user representations and using GRU network to learn short-term user representations through recently browsed articles.

3.2.1 Implementation. To fully understand how the LSTUR works, we implemented a *native* version using the existing implementation in the recommenders repository[7]. Additionally, the ebnerd-benchmarking repository[8] provides an implementation of LSTUR with optimized Dataloader. After some initial testing, we received similar scores, based on the small dataset, for both implementations and opted to use the ebnerd-benchmarking implementation for further hyperparameter tuning and improvements, based on their optimized dataloader. However, it is important to note that implementing the *native* LSTUR, helped a lot in understanding the data and how the algorithm works, as well as how the ebnerd-benchmarking implementation works.

Lastly, we take a look at what we need to provide to LSTUR and how it prepares the data for the algorithm, in a very basic way. First of all the algorithm needs the same data as NRMS: **User History**, **User Behaviors**, **Articles**. These three data sources are now preprocessed and combined with the dataloader. First, we generate Word2Vec embeddings based on the title and sub-title columns. We use the same approach and considerations as for NRMS and leverage Huggingface and a pre-trained model to generate those embeddings. Afterward, we include the user history into the user behaviors and generate labels based on the *Inviewed articles* and *clicked articles*. LSTUR would now need two more things: First, we need a mapping between user IDs and unique IDs and secondly, we need a mapping between the words used in the embeddings and unique IDs. We provide those in the *native* implementation and everything works, however, the ebnerd-benchmarking implementation does this a little bit differently. We still need the user mapping, but the Dataloader replaces the article IDs directly with the corresponding Word2Vec embeddings. Therefore avoid providing it separately and doing it cleaner. Now with all the necessary data provided and basic hyperparameters set, as seen in the next section, training can start!

3.2.2 Optimizations. We need a way to evaluate how well LSTUR is suited for our task. Therefore, we performed some hyper-parameter tuning to assess the potential of this algorithm. We used the following configurations (see table **??**) and tested it with the training- and validation-set of ebnerd-small:

Parameter	**Description**	**Values**
history_size	Size of history considered	10, 50, 100
n_users	Number of users considered	20k, 50k, 70k
title_size	Max title length that considered	10, 50, 100
learning_rate	Learning rate of LSTUR	0.0001, 0.001, 0.01
dropout	Dropout rate of LSTUR	0.1, 0.3, 0.5

Table 2. Parameter Grid for LSTUR

We got very similar results for all configurations varying only with around 4%. The best hyper-parameters were $history\_size = 100$, $n\_users = 70,000$, $title\_size = 50$, $learning\_rate = 0.01$ and $dropout = 0.5$, with an AUC of 0.556. We stopped further optimizations, because our assigned algorithm DKN already outperformes LSTUR at this point and we wanted to focus our optimization efforts on DKN.

[7]https://github.com/recommenders-team/recommenders
[8]https://github.com/ebanalyse/ebnerd-benchmark/

3.3 Deep Knowledge-Aware Network (DKN)

DKN is a recommender system based on a deep-learning neural network that is specifically designed for news recommendation. It incorporates a knowledge graph to combine knowledge about entities with conventional word embeddings. Each news is embedded using 2 embeddings: One by extracting the content of the text and transforming it to tokens, and another by extracting entities from the text (e.g. climate crisis), which are embedded by the similarities between two entities [5].

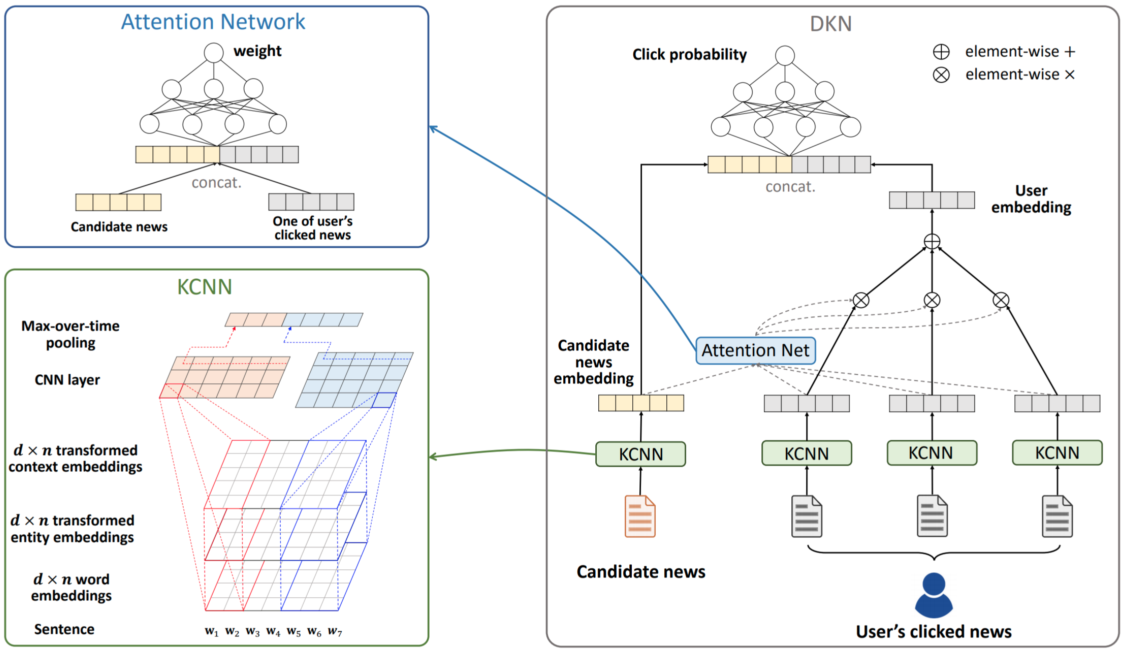


Fig. 1. DKN Architecture [5]

3.3.1 Architecture. The DKN architecture makes use of three different types of embeddings: word embeddings, entity embeddings and context embeddings. Since not every word is part of an entity, entity embeddings are only considered for words that constitute an entity. An entity embedding is a vector representing a single entity in a knowledge graph, whereas a context embedding describes the average of the entity embeddings of all direct neighbor entities of a certain entity in the knowledge graph.

A knowledge-centric convolutional neural network (KCNN) is used to generate news embeddings based on the embeddings of a news text (all embedding vectors for each word of a single candidate news file). Those features are then combined with the user's clicked news embeddings, which have also been transformed in the same manner, using an attention net and passed to a final neural network, which predicts the click probability of the user for the candidate news (cf. Fig. 1).

3.3.2 Input. DKN requires multiple input sources:

Behavior:	contains the labeled data which is trained or predicted, depending on which state the model currently is. For training each recommended (inview) article to a user is in a separate row and has the label 0 (not clicked) or 1 (clicked).
User history:	Each line represents the click history of a user by providing a list of the IDs of previously clicked articles.
Document features:	Contains the word and entity features for each news article. Each line contains the article ID and 2 lists: One for the word embeddings (index to the embedding vector in the embedding file) and one for the entity embeddings.
Embedding files:	Pretrained embeddings for the word embeddings, the entity embeddings and the context embeddings are contained in a 2-dimensional matrix respectively and passed to the model to allow lookups for the embedding vectors of the embedding IDs specified in the document features.

3.3.3 Optimizations. In order to create a DKN model, we transform the train and test data to the expected format. The given data is aggregated: For each impression a list of viewed articles and a list of clicked articles is provided. Our DKN model however expects each article to predict (whether it was clicked or not) in a separate row. Such a transformation has been implemented using Polars' explode method.
The articles file required the most adaptions: DKN requires embeddings from the articles, both based on the content and on the entities, hence we first tokenize the title of each article. For each token we generate GloVe embeddings [4], the entities are ignored for now, as we first need to find a way how retrieve those entities.
All modifications on the data are done using Polars Dataframes and the results are saved in a TSV format, those files are then read by the DKN implementation during training, evaluation and prediction.

Transformation to the format expected by the DKN implementation works now, but we are facing issues when converting the larger data sources, namely the large dataset and the test set. Those issues are caused by high memory usage, which in turn is caused by loading the file in memory as a whole. To limit memory usage, we switch to Polars' streaming API wherever possible, which allows us to read small chunks of the file in multiple steps instead of first loading the whole file into memory. After those optimizations, the transformation of all data sizes is working and we are ready to start our optimizations.

After bringing the data into the right format and optimizing the transformation pipeline, we noticed that there were a lot of words in the document features file that did not have a corresponding word embedding. The default implementation of DKN we built our implementation upon made use of GloVe embeddings [4] , which try to represent the most common words based on a Wikipedia corpus in a vector space. The *ACM RecSys 2024 Challenge* dataset consists of Danish news articles, which suggests that Danish words are underrepresented in GloVe embeddings. For this reason, we decided to make use of Word2Vec embeddings specifically trained on a Danish corpus provided by the Language Technology Group at the University of Oslo[9]. By using these Danish embeddings we managed to reduce the number of words without an according embedding significantly and could observe a big improvement in the models' performance.

We were unable to re-use the entity embeddings used by the base implementation. Thus, we made use of a Python library called DaCy[10], which is a Danish wrapper of the popular NLP library SpaCy[11], providing resources for Danish natural language processing, in order to perform named entity recognition (NER) in combination with named entity linking (NEL) on our dataset. NER describes the process of classifying named entities from a text passage, while NEL

[9]http://vectors.nlpl.eu/repository/
[10]https://github.com/centre-for-humanities-computing/DaCy
[11]https://spacy.io/

links said entities to entries in a knowledge graph like Wikidata[12]. This process provided us with IDs of entities in the Wikidata knowledge graph based on the news texts in the dataset. We then used an API providing embeddings for Wikidata entities called Wembedder[13] to retrieve our final entity embeddings.

Lastly, we generated the context embeddings by using the Wikidata API[14] to retrieve the direct neighbors of an entity, looking up their embeddings as described before and calculating their average value.

3.4 Training & Evaluation

To estimate required training and prediction times, we used DKN, as it was the first model which could handle all given dataset sizes. The performance metrics of Table 3 were created by running on a separate machine utilizing a GTX 1070 and 32GB of RAM, it was not done on the provided GPU cluster in order to save computing resources as advised by the lecture team. On the cluster we achieved approximately a 40% reduction in runtime due to the more performant hardware.

Dataset	History Size	Epochs	AUC	Mean MRR	NDCG@5	NDCG@10	Runtime (h:m)
small	50	2	0.556	0.3386	0.379	0.4579	1:42
small	5	10	0.545	0.336	0.3734	0.4546	1:07
small	50	10	0.6196	0.4146	0.4557	0.5228	8:22
large	5	2	0.555	0.343	0.383	0.462	8:52
small (improved)	50	10	0.6956	0.4934	0.5387	0.5919	12:00
large (improved)	50	2	0.6061	0.4012	0.4444	0.5127	> 24:00

Table 3. Runtimes for base DKN model

Due to the excessive runtime of the large dataset (c.f. Table 3), we skipped additionally planned benchmarks on the large dataset and chose to optimize and evaluate mainly on the small dataset.

We performed hyperparamter optimization for both the baseline model NMRS and the chosen model LSTUR. For the hyper-opt, we used the demo dataset and validated it on the demo validation set. Initially, we conducted a grid search for the following hyperparamters:

- **History Size**: Determines the amount of user history considered for the recommendations, impacting the model's ability to learn user preferences over time.
- **Title Size**: Refers to the title size we want to consider in the prediction process. As most of our algorithms are content-based recommenders, this parameter may have high influence on their performances.

In Figure 2, we observe that a higher history size improves the accuracy's. Additionally, a higher title size results in better performance, except when increasing the title size beyond 50 (e.g. to 100) resulting into a decline in accuracy, especially for NRMS. A possible reason for this could be that with a very large title size, the model might struggle with overfitting or fail to learn useful item features due to increased complexity.

[12]https://www.wikidata.org/
[13]https://wembedder.toolforge.org/
[14]https://www.wikidata.org/wiki/Wikidata:REST_API

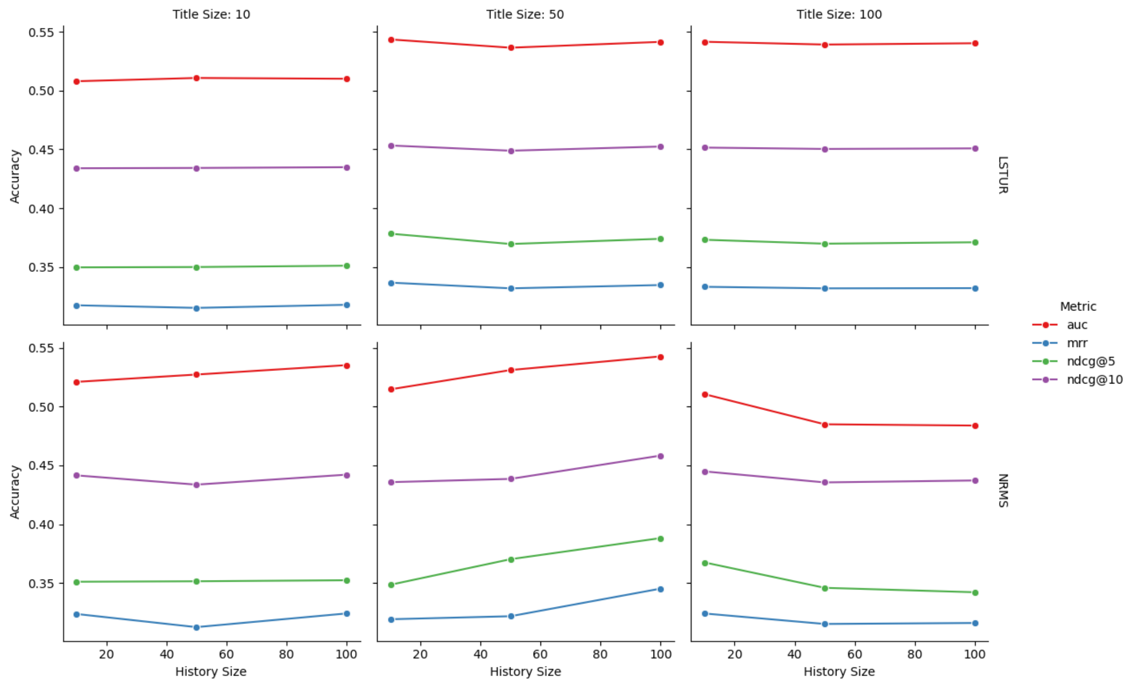


Fig. 2. This FacetGrid illustrates the performance of the LSTUR and NRMS algorithms across different history and title sizes trained and evaluated on the small dataset. Each subplot represents a title size, with lines indicating performance metrics (AUC, MRR, NDCG@5, NDCG@10) over varying history sizes. The results indicate that larger title sizes do not necessarily improve accuracy for NRMS, while there is a correlation between accuracy and history size.

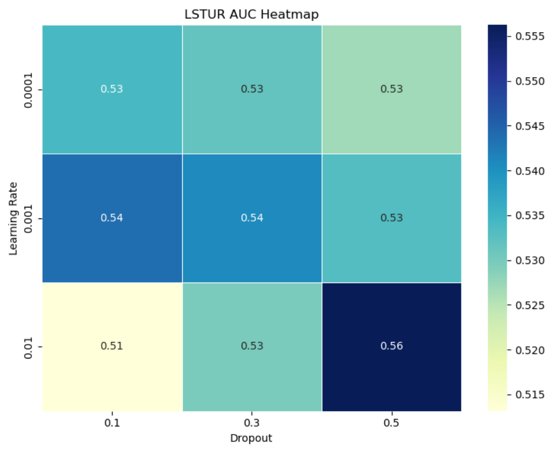
In Figure 3, we see that for LSTUR, a high dropout of 0.5 and a high learning rate of 0.1 works well, while for NRMS, the best hyperparamters are a moderate learning rate and dropout of 0.001 and 0.3. A reason for the LSTUR improvement using higher dropout could be that it helps prevent overfitting by making the model more robust and forcing it to learn more general features.

4 Results

To compare the performance of all the algorithms we implemented, we chose to train them in a similar fashion, by using a history size of 50 and training for a maximum of 10 epochs with early stopping enabled. As described earlier, after some simple hyperparameter tuning, we saw that the DKN model was able to achieve better performance than the other algorithms across all evaluation metrics used, thus we decided to optimize it further.

Model	AUC	Mean MRR	NDCG@5	NDCG@10
NRMS	0.5472	0.3363	0.3790	0.4560
LSTUR	0.5562	0.3450	0.3847	0.4616
DKN	0.6196	0.4146	0.4557	0.5228
Improved DKN	0.6956	0.4934	0.5387	0.5919

Table 4. Results of different models evaluated on small dataset



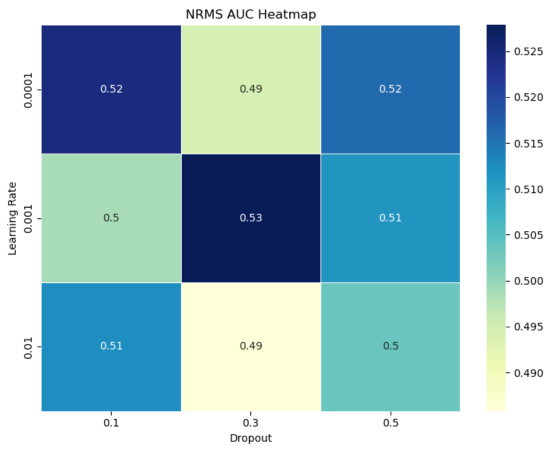


Fig. 3. AUC Heatmaps for Different Dropout and Learning Rates: The left heatmap shows the AUC performance of the LSTUR model, and the right heatmap shows the AUC performance of the NRMS model. For LSTUR, a higher dropout rate (0.5) and learning rate (0.01) yield the best performance. For NRMS, a moderate learning rate (0.001) and dropout rate (0.3) achieve the best results.

In Table 4 we summarized the performance of the LSTUR, NRMS and the DKN models trained on the small *ACM RecSys 2024 Challenge* dataset and evaluated them using the AUC, Mean MRR, NDCG@5 and NDCG@10 metrics on the small test set. Additionally, we include the performance of the improved DKN model with Danish Word2Vec Embeddings, Entity Embeddings and Context Embeddings.

The NRMS utiltizes mult-head self-attention mechanism, which allows it to capture complex relationship by directly directly training on textual features, which makes it optimal as a typical and easy to use content-based filtering recommenders system. In order to properly recommend it however needs lots of training data and content to learn from.

The LSTUR model excels at understanding user behavior over time by combining long-term and short-term user interests. This dual representation allows the model to adapt to both the user's stable interests and recent activity, making it highly effective for dynamic news recommendation scenarios. LSTUR is particularly well-suited for platforms where users have frequent and varied interactions, as it can leverage both historical and recent behaviors to provide personalized recommendations. However, LSTUR's reliance on extensive user interaction data can be a problem for new users with little interaction history. Additionally, it requires significant computational resources, making it challenging for large-scale applications.

For the DKN model, we were able to achieve quite good performance out of the box without improvements regarding the word embeddings and without the use of any entity or context embeddings. DKN is built for news recommendations and thus it is no surprise that it works quite well at exactly that task, and even better if the right improvements are made to its inputs. However, this performance comes at a cost - complexity. The inputs it requires are quit sophisticated and quite cumbersome to generate. The complexity of its architecture is also reflected in its runtime, which was quite high in our experience and only increased further with increasing parameters such as the history size considered.

5 Conclusion

We implemented 3 models to solve the Ekstra Bladet RecSys Challenge:

NRMS: Used as baseline and provided the worst performance of the 3 models.

LSTUR: Showed a better performance after hyperparemter tuning. Its architecture is a good fit for news recommendation as it combines both the long term interests, while also keeping a short term memory of currently relevant topics.

DKN: Outperformed both NRMS and LSTUR, achieving the highest AUC (main performance criterium) among the three models. Its integration of knowledge graphs and entity embeddings provided an architecture specifically designed for new recommendation, which showed in the good results. Due to the good performance, we focused our main attention and optimized this model, e.g. by using language specific word embeddings, which increased the AUC even further.

To achieve those results we applied 3 Methodologies: (1) Preparing and cleaning the data in order to make it usable by the model. (2) Implementing and improving the algorithm. (3) Hyperparameter tuning if performance increases by optimizing model parameters seem likely.

We have seen that news recommendations is an especially hard topic for recommender systems due to the short lifetime of a single news. By implementing otherwise less utilized aspects (content based, entity embeddings), we were able to improve our models substantially compared to the baseline. However, adding additional embeddings and information comes at the cost of high runtimes, which we had to face throughout the challenge.

References

[1] Mingxiao An, Fangzhao Wu, Chuhan Wu, Kun Zhang, Zheng Liu, and Xing Xie. 2019. Neural news recommendation with long- and short-term user representations. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Anna Korhonen, David Traum, and Lluís Màrquez, (Eds.) Association for Computational Linguistics, Florence, Italy, (July 2019), 336–345. DOI: 10.18653/v1/P19-1033.

[2] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. (2014). https://arxiv.org/abs/1406.1078 arXiv: 1406.1078 [cs.CL].

[3] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: a highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*. I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, (Eds.) Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[4] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 1532–1543. http://www.aclweb.org/anthology/D14-1162.

[5] Hongwei Wang, Fuzheng Zhang, Xing Xie, and Minyi Guo. 2018. Dkn: deep knowledge-aware network for news recommendation. (2018). arXiv: 1801.08284.

[6] Chuhan Wu, Fangzhao Wu, Suyu Ge, Tao Qi, Yongfeng Huang, and Xing Xie. 2019. Neural news recommendation with multi-head self-attention. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, (Eds.) Association for Computational Linguistics, Hong Kong, China, (Nov. 2019), 6389–6394. DOI: 10.18653/v1/D19-1671.