Winning Amazon KDD Cup'23

Chris Deotte
NVIDIA
USA
cdeotte@nvidia.com

Kazuki Onodera
NVIDIA
Japan
konodera@nvidia.com

Jean-François Puget
NVIDIA
France
jpuget@nvidia.com

Benedikt Schifferer
NVIDIA
Germany
bschifferer@nvidia.com

Gilberto Titericz
NVIDIA
Brazil
gtitericz@nvidia.com

ABSTRACT

This paper describes the winning solutions of all tasks in Amazon KDD Cup '23 from the NVIDIA MERLIN team[1]. The challenge was to build a multilingual recommendation system. From each user, we are given a history of item interactions and we need to predict the next item interaction. Our solution for tasks 1 and 2 is a pipeline of candidate generation, reranking, and ensemble. For candidate generation we leveraged statistical models, representation learning with embedding loss, pre-trained language models, multi-task learning with transformers, and more. Candidate sets were merged and ranked using gradient boosting (XGBoost and CatBoost) to maximize MRR score. Task 3 solution is based on multiple classifiers to maximize BLEU score.

ACM Reference Format:
Chris Deotte, Kazuki Onodera, Jean-François Puget, Benedikt Schifferer, and Gilberto Titericz. 2023. Winning Amazon KDD Cup'23. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Modelling customer shopping intentions is crucial for e-commerce stores, as it directly impacts user experience and engagement. Accurately understanding what a customer is searching for, such as whether they are looking for electronics or groceries with the search query "apple", is essential for providing personalized recommendations. Session-based recommendation, which utilizes customer session data to predict their next purchase, has become increasingly popular with the development of data mining and machine learning techniques. However, few studies have explored session-based recommendation under real-world multilingual and imbalanced scenarios.

To address this gap, Amazon has organized the KDD Cup 2023 challenge with 3 different tasks exploring 3 different angles of multilingual product recommendation [JML+23]. This competition setup and the dataset it uses are described in section 2.

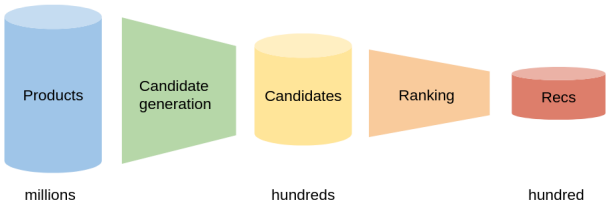

Figure 1: Our pipeline for task 1 and task 2.

A team from NVIDIA (the authors of this paper) entered the competition and won all 3 tasks. This paper describe the team solutions[2] to all 3 tasks.

Our solution for the first 2 tasks are very similar and can be described jointly, except for some transfer learning specific to task 2 (section 7). Our solution is a multi-stage pipeline (see figure 1) with

(1) Candidate Generation
(2) Ranking Candidates,
(3) Weighted Average Ensemble.

We describe each of these steps in sections 3, 4, and 5.

Our solution to task 3 is based on multiple classifiers to maximize BLEU score. It is described in details in section 6

We then conclude the paper with a discussion of how we used transfer learning methods to deal with underrepresented languages, followed by a summary of our main contributions.

2 AMAZON KDD CUP'23

2.1 Dataset

For the challenge Amazon created the "Multilingual Shopping Session Dataset" [JML+23], consisting of millions of user sessions from six different locales, where the major languages of products are English, German, Japanese, French, Italian, and Spanish. The dataset is imbalanced, with fewer French, Italian, and Spanish products than English, German, and Japanese.

[1]All authors have an equal contribution.

[2]We provide all the code of the solutions at: https://gitlab.aicrowd.com/BenediktSchifferer/kdd2023cup_nvidiamerlin . This git repository also contains description files and readme files that contain details that we could not include in the paper for the sake of space.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Language (Locale)	Sessions	Products (ASINs)
German (DE)	1111416	513811
Japanese (JP)	979119	389888
English (UK)	1182181	494409
Spanish (ES)	89047	41341
French (FR)	117561	43033
Italian (IT)	126925	48788

Table 1: Dataset statistics.

The dataset consists of two main components: user sessions and product attributes. User sessions are a list of products that a user has engaged with in chronological order, while product attributes include various details like product title, price in local currency, brand, colour, and description.

Table 1 summarizes the dataset statistics, including the number of sessions, interactions, products, and average session length.

In addition, participants were provided with product attribute data, including textual data for title, description, brand, color, material, etc. Price in local currency was also provided. User ratings (stars) where not provided. See [JML+23] for more details on the data.

2.2 The Competition Tasks

The challenge had 3 tasks, hosted on AIcrowd:

(1) predicting the next engaged product for sessions from English, German, and Japanese,
(2) predicting the next engaged product for sessions from French, Italian, and Spanish, where transfer learning techniques are encouraged,
(3) predicting the title for the next engaged product.

Task 1 aims to predict the next product that a customer is likely to engage with, given their session data and the attributes of each product. For each test session, the participants had to predict 100 product IDs (ASINs) that are most likely to be engaged, based on historical engagements in the session.

The goal of Task 2 was similar to Task 1, while the test set is constructed from French, Italian, and Spanish. Task 2 focuses on the performance on these three underrepresented languages.

The evaluation metric for Task 1 and Task 2 was Mean Reciprocal Rank (MRR). The evaluation metric for task 3 was the BLEU score between the predicted title and the actual title.

3 CANDIDATE GENERATION

3.1 Representation learning

An effective method to generate candidates is to use latent vectors (a.k.a. embeddings) for sessions and products by following steps:

- compute product embeddings
- compute session embeddings
- predict next items on test data using a nearest neighbor search on test session embeddings and product embeddings

An early way to compute these embeddings was collaborative filtering with matrix factorization. This method is based on past interactions only. Here we had more information than just past interactions. In particular we had a set of text data for each product. We leveraged this to compute embeddings from these texts, then used a k nearest neighbor (KNN) search, with k ranging from 50 to 200. We use cosine similarity to measure the similarity of two embeddings P, and S (S and P being two vectors):

$$sim(S, P) = \frac{S.P}{||S||.||P||}$$

These candidates and their cosine similarity with the session embedding are then fed into a reranker (see section 4).

We use various ways to compute embeddings.

3.2 Using Pretrained LLMs

A very effective way to get good embeddings from text is to use pretrained transformers and use their last layer activation before the classification head. We used 7 different multi lingual LLMs. Embeddings were computed on the concatenation of texts for locale, title, brand, color, price, size, model and material. In order to add some diversity, texts were truncated (e.g. taking only first 80 tokens for title) for some of the pipelines. For some models we also added a textual representation for the price. Prices were normalized across countries, using the fact that the same product (ASIN) can be present in more than one country. Some of the candidate lists were merged and top 200 candidates were kept. Before merging the similarities were weighted in order to take into account the difference in performance (recall or MRR) of each LLM. As a result we got several candidate lists per session, with similarity scores.

3.3 Using Contrastive Learning with CNNs

We devised a new method to compute embeddings which is way more powerful than collaborative filtering on this dataset. It has 3 steps:

- compute initial product embeddings from product titles
- create a model that takes as input prev items embeddings and outputs a session embedding
- train the product embeddings by maximizing the cosine similarity between session embeddings and sessions next items, while minimizing the cosine similarity between session embeddings and random products embeddings
- predict next items on test data using a nearest neighbor search on test session embeddings and product embeddings

We started with a simple model that predicts the next item embedding from the last 3 prev items embeddings a shown in figure 2. The model is trained to do both:

- maximize the similarity between each predicted embedding and the embedding of its following item
- minimize the similarity between each predicted embedding and the embedding of random products

When training, back propagation updates the product embeddings in two ways:

- a direct way when the product is used as a positive or a negative sample in the loss,
- an indirect way through the model when the product appears as a prev item of the session.

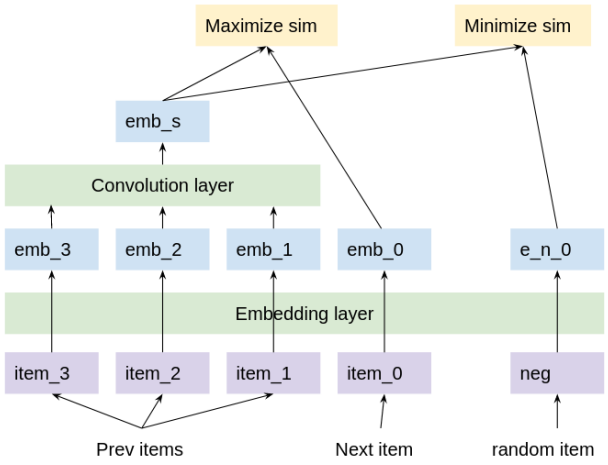

Figure 2: A simple CNN. which takes the embeddings of the last 3 prev items, then applies a convolution to them to get a session embedding. The output is then fed into a loss that maximizes the similarity with the next item embedding, and minimizes the similarity with random items

A good improvement was obtained by initializing the model embedding table from the embeddings generated by a pretrained LLM. Best results were obtained using xlrm-roberta finetuned on sentence pairs: sentence-transformers/stsb-xlm-r-multilingual This model creates embeddings of length 768.

We tried adding more embeddings. What worked best was to also use description embeddings using the same model. We added 0.3 times the description embeddings to the title embedding.

Another refinement was to compute embeddings for the brand of each product. We use embeddings of length 16 that are concatenated to the title + description embeddings. The brand embeddings were initialized randomly.

We then generalized this model to apply the convolution to the last k prev items, which produces k - 2 embeddings with a convolution of size 3. Each of these embeddings is then passed to the loss to maximize its similarity with the item immediately following it, and at the same time minimizing the similarity with random items. We used convolutions of size 3 or 4 applied to the last 16 prev items. When using a convolution of size 4 we get 13 predicted embeddings. When we use a convolution of size 3 we get 14 predicted embeddings.

For the sake if simplicity we show the model with k=4 instead of k=16 in figure 3 .

We used a cosine embedding loss with margin, similar to Pytorch torch.nn.CosineEmbeddingLoss.

Given a predicted session embedding S, and a positive product embedding P, the loss is 1 minus the cosine similarity, i.e:

$$loss_{+}(S,P) = 1 - \frac{S.P}{||S||.||P||}$$

Given a predicted session embedding S, and a negative product embedding P, the loss is the cosine similarity if it is above a given margin, i.e.:

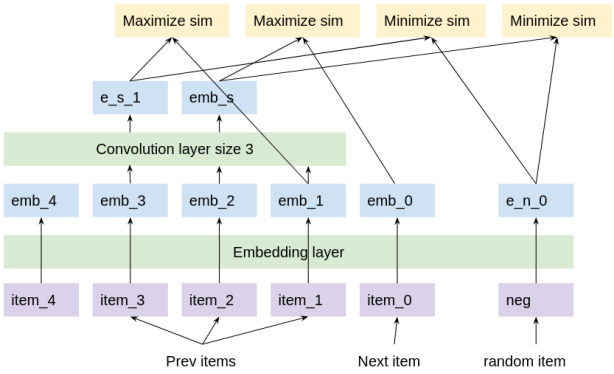

Figure 3: CNN model with k=4. The convolution produces 2 embeddings. The last one is the session embedding. We use 4096 random samples for the negative loss for each positive loss sample (only one is depicted in the figure.)

$$loss_{-}(S,P,margin) = max(0, \frac{S.P}{||S||.||P||} - margin)$$

We used high margin values of 0.65 or more. Both the very large number of negative samples and the large margin value were required to get good results. For a batch size of 256, our model outputs 14 embeddings per sample, and uses 4096 random samples as negatives for each predicted embedding, which makes a total of 256 * 4096 * 14 = 14M negative samples per batch.

The CNN model works pretty well except for long sessions as it ignores most of their prev items. We therefore added a second model that takes the 8 most frequent prev items and applies a single convolution of size 8 to their embeddings. The embeddings are concatenated with the item frequency before the convolution. The output of this frequency based model is added to the last output of the previous model to yield the final session embedding. The frequency based model improves score by about 0.003.

We trained a model for each of: locale DE, locale UK, locale JP, and all task 2 locales at once. When we sample negatives we restrict them to have the same locale(s) as the model. The models are implemented with Pytorch 2.0.

The test predictions could be used directly. They yield a MRR of 0.37804 on task 1 and 0.42959 on task 2. These are good scores but not competitive with top scores. Reason is that this method does not learn any per session or per product feature. For instance it cannot model products popularity. However, the candidate products and their similarity with sessions are very good features for the reranker models, see section 4.

3.4 Using MLM with Transformers

Another method is to use transformer layer instead of convolutional layer to process the input sequence of item ids to generate a session embedding as visualized in Fig 4.

The input is the sequence of prev items with item id and its item features, such as brand, color, size, model, material, author and pre-trained BERT embeddings. The categorical inputs are fed through an embedding layers. The embeddings are concatenated

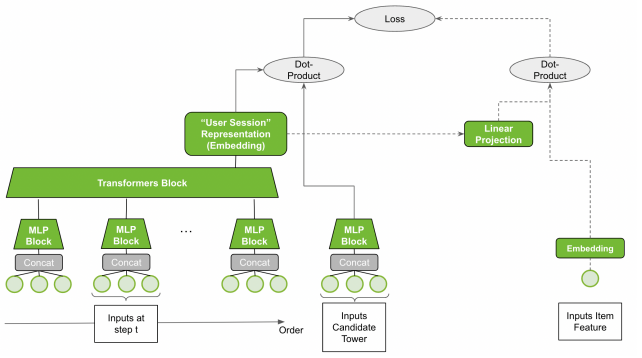

Figure 4: The neural network uses a transformer-layer to process the input sequence of prev items with item id, its item features and the pre-trained BERT embeddings. One option is to use Multi-Task Learning to predict the masked item's features.

and processed by a 1-layer MLP tower without activation function. A XLNet architecture is used to process the sequence and generate an embedding to represent the session (session embedding). An item-tower, similar to a two-tower architecture, is used to generate embeddings for each candidate. The item-tower shares the weights of the embeddings and 1-layer MLP tower with the input processing of the transformer. Finally, weight-tying is applied to the session embedding and item embedding as proposed in [SDP+21] to get a score for the likelihood that this candidate is the target for the sessions.

The models are inspired by language modelling as analyzed by Transformers4Rec [dSPMRL+21]. The library provides many architectures and training strategies. Masked Language Modelling (MLM) is used for training. Items are masked in the input sequence with probability 0.35. As the item catalog is upto 500k items per language, training the models over all items is inefficient. Sampled Softmax with negative sampling is used to train the model. The negative candidates are generated by in-batch sampling to generate hard negatives and uniform negative sampling (4x the batch size of 1024) to avoid popularity items get penalized. During inference all items are predicted. The models are trained per language.

The prediction should be used in the ranking stage, therefore, the training and test dataset needs to be predicted. To avoid leakage from the test data into the training predictions, each model is trained twice:

- It is trained only with training data to predict session in the training dataset.
- It is trained with training and test data to predict sessions in test dataset - using only prev items of the test dataset.

In the 2nd case, the training process is to iterate over training dataset, first and then over the test dataset for each epoch.

The training dataset contains prev items and next item. One variation are models trained only on prev items, these do not require out-of-fold (OOF) predictions. Models, which uses next item data, concatenates next item with the prev items as MLM Masking samples from the input sequence. To avoid data leakage for the ranker, the training dataset was randomly split into 5-folds. Next items are not weighted differently in the loss, which could be investigated in future research.

Another option is to train models with a Multi-Task objective (dotted lines in Fig 4). In addition to predicting the item id, the model predicts the item features brand, color, size, model, material, author. For each item feature, the session embedding is fed through a linear layer without activation function and the dot-product between the output and the respective embedding table weights are calculated. The item id loss was weighted by 5x .

In total, we trained four transformers-based models:

- Single-Task using only prev item
- Multi-Task using only prev item
- Single-Task using only prev item + next item
- Multi-Task using only prev item + next item

3.5 Co-visitation Matrices

We also generated candidates via a quite powerful technique called co-visitation matrices.

From each session history, we consider every combination of two items (from their complete list of history items). These are pairs of items. Each pair implies that when a user interacts with Item A then they are likely to also interact with Item B. By using all 3.6 million session's histories we count up every occurrence of each pair of items and we keep the most frequent ones.

There are many options to calculate the score. We will describe some examples.

A simple method is to count the number of pairs.

$$sim_{AB} = \sum_{A,B \in S, A \neq B} 1$$

Another option is to restrict the pairs that B appears after A

$$sim_{AB} = \sum_{A,B \in S, A \neq B, Pos_A < Pos_B} 1$$

Another restriction could be that the pair A, B has to appear in a certain window.

$$sim_{AB,k} = \sum_{A,B \in s, A \neq B, |Pos_A - Pos_B| <= k} 1, k \in \{1, 3, 5\}$$

Another option is to weight the pairs depending on the position differences in the sequence and the total length of the session [ZZW22]

$$sim_{AB} = \sum_{A,B \in s, A \neq B} \frac{1}{log(|Pos_A - Pos_B| + 2} * \frac{1}{log(SessionLength + 2)}$$

After executing this step, for each of 1.4 million items, we have a top 100 list of which additional items are frequently paired with it.

To generate 100 items that a user will be interested in, we iterate through each item in a users' history. For each item, we add the top100 list of corresponding items that are paired with the history item. After iteration, we have many items that appear multiple times. We take the 100 most common items, and these are 100 items that a user will be interested in.

Another variant is to only consider the last visited item instead of all the visited items. This yields a different set of candidates. We also used the last two items, which yields yet another set of candidates.

Another variant is obtained by including the next item in the co-visitaion matrix computation. Similarly, this yields yet another set of candidates.

We generated many combination, likely in the 50-100, between different options to calculate the similarity matrix and applying it to the user. Each output provides a score for each (user, item) pair, describing which item is most likely the next item for a user, which can be used in the ranking stage.

3.6 Swing Similarity

Similar to co-visitation matrices, Swing similarity can be calculated as described in [ZZW22]. The swing algorithm captures can capture the inner structure of the user-item behavior graph. The authors describe it to be more stable than traditional CF approaches.

The Swing similarities are applied in a way similar to CoVisitation Matrices to each session. This yields a new set of candidates.

4 RERANKING

4.1 Reranking with Binary Target

For each of the candidate generator describe above we get 100 candidates per user. We create a data frame with 360 million rows. For each user, there are 100 rows. The first column has the user id and the second column has the item id. Depending on the candidate generator we can have additional rows, for instance the cosine similarity and the rank in the KNN search, or the frequency in the co-visitation matrix used. Additional features may be added depending in the generator, see 4.2. Several such data frames can be merged on the user. This yields an even larger data frame, with more than 1000 candidates per user. To speedup computation we sometimes split these data frame per locale.

Lastly we create a target column. We set target = 1 if the candidate item is the next item that the user interacts with and we set target = 0 if the candidate item is not the next item that the user interacts with. Using this data frame, we train gradient boosting models (XGBoost [CG16] and CatBoost [PGV+19]) using features except of the target. Our models are binary classifiers which predicts the probability that the candidate item is the next item that the user interacts with.

The same dataset pipeline is applied to test data, and the classifier is applied to the end data frame. This yields probabilities for each session candidates and we retain the top 100 most probable ones as the final prediction.

We used either binary log loss or pairwise ranking loss to train these models. Using two different loss adds some diversity which helps when ensembling.

The training data frame contains 360 million rows and about 150 features (approximately 250GB in memory). In order to train GBT in a reasonable amount of time we used GPUs. A challenge was to train a single GBT model using all this data at once. We used dask [das23] and dask_cudf [cud23] to train with 100% data 3000 trees of XGBoost using 8x NVIDIA V100 GPUs with each 32GB memory and were able to train a model in about 1h. Also using 8xV100, we trained 80% data and 3000 trees using CatBoost.

4.2 Feature Engineering

A number of features were created in addition to what candidate generators provide out of the box. These include items features, user features, and user - product interaction features.

For some features we trained additional models, a GRU, to predict the next item brand from the prev items brands. The predicted probability of the next brand is added as a feature.

The cosine similarities and rank from KNN generators are also used as features

Here are few powerful features:

- co-visitation matrix count from main covisit matrix
- ranking from co-visitation matrix generator
- cosine similarity from CNN model embeddings
- co-visitation matrix using prev-2 prev-1 next items only
- gru to predict brand. Then probability of candidate item's brand
- co-visitation matrix using brands only to predict brand
- co-visitation matrix count using last user history item only
- price of the last prev item divided price candidate item
- gru to predict model. Then probability of candidate item's model
- cosine similarity between candidate item title and last prev item title using pretrained LLMs.
- target encoding using out-of-fold count of candidate item appearing in train next items
- count of unique users engaging with each candidate item

We used hundreds of features in the solution. They are listed in the pdf files and readme files of our code repository.

5 ENSEMBLE

The description above produces submissions for tasks 1 and 2. For each submission, and from each user we have 100 predictions and their corresponding probability of being the next item that a user interacts with.

Given we used two different losses for training rerankers, we needed to invent a way to ensemble predictions from ranking loss (which are not probabilities) with predictions from binary logloss (which are probabilities). We took the probability distribution from a reranker trained with log loss, and recalibrated other rerankers scores to match its probability distribution. We simply sorted the other reranker session outputs by score, then replaced the score by the sorted probabilities.

We outer merge all the team members predictions together. When more than one submission predicts the same item for the same user, we use a weighted average of the probabilities from each of submission. When a submission predicts an item which is not predicted by other submissions then we assume the other submissions predict a probability of 1e-6 and compute a weighted average.

We discovered that increasing the scores of the items that appear in test data before taking the top 100 improved the submission score. We also found that increasing further the scores of items that only appear in test data improved the submission score further.

Before applying mutlipliers we converted probabilities to odds. Best multipliers we used for task 1 were 7 and 10. Best mutipliers we used for task 2 were 10 and 15.

Afterward, we sorted all the new scores and take the top 100 for each user. This is how we created our final submissions. We got a MRR of 0.41188 in task 1 and a MRR of 0.46845 in task 2, securing 1st place in both tasks.

6 TASK 3

Task 3 challenges us to predict a title for each users' next item interaction. We discovered that submitting a user's most recent history item's title as the prediction achieves a good submission score of BLEU = 0.26553. To improve this, we trained a classifier to choose between using a user's most recent history item title versus second most recent history item title. Inputs for this model includes predictions generated by models trained on tasks 1 and 2, especially the cosine similarities form the CNN and Transformer models. Top10 candidates were pick from each model.

Additional features were generated. For each candidate and last 2 titles in history:

- differences: len(candidates title) - len(last tile)
- differences: len(candidates title) - len(second to last tile)
- lenghts of title and candidates
- number of words intersection between each candidate and last 2 titles
- Average number of words per title for each candidate list: CNN and Transformer.
- Difference of number of words of last 2 titles and the previous average of candidates.
- BLEU score between each CNN and Transformer candidates and last 2 titles in history.

Training dataset was sampled to contain the same distribution of locales as the test dataset. A RandomForest algorithm was chosen to avoid any kind of overfit. This improved last title submission score by +0.0012.

Next, since BLEU score is based on n-gram precision (where less words are better if we avoid a brevity penalty), we train a second classifier which predicts whether we should remove the last word from the output of our first classifier. For about half of predictions, we remove the last word without incurring a brevity penalty. This improved submission score by +0.0048. Using all train data titles, we created targets which equal 1 if removing the last word improves BLEU and 0 if removing the last word does not improve BLEU. We then use the title text as input and the new targets to train XLM-Roberta-base. Using the predictions from this second classifier, we remove the last word from 50% of the prediction titles. With this mixed approach we got a BLEU score of 0.27152 and won task 3.

7 TRANSFER LEARNING

Task 2 had way less data than task 1. Competitor hosts encouraged participants to use transfer learning techniques to mitigate the too little training data issue. Here is what we did:

- create co-visitation matrices for task 2 languages by using users' histories that exist in both tasks languages.
- represent items with multilingual LLM (large language models) embeddings.
- init embeddings in CNN model with pretrained embedidngs from a multilingual LLM (large language models)
- use item features from task 1 languages as item features for task 2 languages in tabular dataframe for reranker
- create user-item interaction features by transferring user-item patterns learned from task 1 languages to task 2 languages.
- train task 2 reranker using user-item dataframe rows from task 1

These also improved our task 3 solution as all languages are used in that task, including task 2 languages.

8 CONCLUSION

The Amazon KDDCup'23 competition was a unique challenge due to its multi lingual data and also due to some languages being underrepresented. We have presented how we addressed these challenges successfully given our solutions achieved the best scores in all 3 tasks of the competition. Our solution for tasks 1 and 2 is a pipeline of candidate generation, reranking, and ensemble. For candidate generation we leveraged statistical models, representation learning with embedding loss, pre-trained language models, multi-task learning with transformers, and more. Candidate sets were merged and ranked using gradient boosting (XGBoost and CatBoost) to maximize MRR score. Task 3 solution is based on multiple classifiers to maximize BLEU score.

REFERENCES

[CG16] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery.

[cud23] cudf - gpu dataframes. https://github.com/rapidsai/cudf, 2023. Accessed: July 20, 2023.

[das23] Dask: Scalable analytics in python. https://dask.org/, 2023. Accessed: July 20th, 2023.

[dSPMRL+21] Gabriel de Souza Pereira Moreira, Sara Rabhi, Jeong Min Lee, Ronay Ak, and Even Oldridge. Transformers4rec: Bridging the gap between nlp and sequential / session-based recommendation. In *Proceedings of the 15th ACM Conference on Recommender Systems*, RecSys '21, page 143–153, New York, NY, USA, 2021. Association for Computing Machinery.

[JML+23] Wei Jin, Haitao Mao, Zheng Li, Haoming Jiang, Chen Luo, Hongzhi Wen, Haoyu Han, Hanqing Lu, Zhengyang Wang, Ruirui Li, Zhen Li, Monica Xiao Cheng, Rahul Goutam, Haiyang Zhang, Karthik Subbian, Suhang Wang, Yizhou Sun, Jiliang Tang, Bing Yin, and Xianfeng Tang. Amazon-m2: A multilingual multi-locale shopping session dataset for recommendation and text generation. 2023.

[PGV+19] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features, 2019.

[SDP+21] Benedikt Schifferer, Chris Deotte, Jean-Francois Puget, Gabriel de Souza Pereira Moreira, Gilberto Titericz, Jiwei Liu, and Ronay Ak. Using deep learning to win the booking.com wsdm webtour21 challenge on sequential recommendations. In *Proceedings of the Workshop on Web Tourism (WebTour) co-located with the 14th ACM International WSDM Conference (WSDM 2021)*, volume Vol-2855. [Sl]: CEUR, 2021.

[ZZW22] Zzh, Wei Zhang, and Wentao. Industrial solution in fashion-domain recommendation by an efficient pipeline using gnn and lightgbm. In *Proceedings of the Recommender Systems Challenge 2022*, RecSysChallenge '22, page 45–49, New York, NY, USA, 2022. Association for Computing Machinery.

9 REPRODUCIBILITY

All the code for our solution is available at:
https://gitlab.aicrowd.com/BenediktSchifferer/kdd2023cup_nvidiamerlin .

We describe in detail how to run the code in the readme files in that git repository. We also describe the execution environment.

The top level directory provides instruction on how to run each which directory code in which order. We provide detailed instruction on how to run the code in the readme file for each sub directory.

We used models downloaded from Huggingface Hub at the time of the competition. These models are:

- bert-base-multilingual-uncased
- xlm-roberta-base
- xlm-roberta-large
- sentence-transformers/allenai-specter
- sentence-transformers/distiluse-base-multilingual-cased-v
- bert-base-multilingual-uncased
- sentence-transformers/stsb-xlm-r-multilingual
- sentence-transformers/clip-ViT-B-32-multilingual-v1

The code was run on DGX V100 workstation or on NVIDIA cluster nodes with up to 8 V100 GPU (DGX-1 machines). The CNN model in particular was quite compute intense, with more than 24 hours per language on a 8xV100 node.