Fantlab Recommender System

A. Nepomnyashchy, Yu. Popov

May 2022

Abstract

This project is an attempt to build a recommender system for a science fiction and fantasy literature resource fantlab.ru involving NLP techniques. The code used in project you can find here: `https://github.com/yupopov/fantlab-recommender-system`.

1 Introduction

Recommender systems are ubiquitous these days, every single popular content resource has its own. Pros of using them are self-evident. Mainly, there are three types of recommender systems: collaborative, content and hybrid. Collaborative filtering predicts the interests of a single user by collecting preferences information from many users with the similar tastes, they are rather fast, but not so strong. Content-based systems predict what a single user will like by using attributes of the user, such as age and location (user features) and attributes of the content, such as tags (item features). Content-based systems are much stronger than collaborative, but also significantly slower, especially at large amount of content. Hybrid model is the most effective way to build a recommendation system nowadays. Firstly, it chooses n most suitable items for a single user using a collaborative model (generally $n \leq 1000$) and then re-ranks these items with the content model.

Recommender systems today are almost entirely machine learning models that train on user feedback. User feedback comes in two very distinct forms:

- **Explicit feedback**, that requires conscious effort from user (such as like/dislike buttons, ratings on scale from 1 to 10, user reviews and so on).
- **Implicit feedback**, that can be collected from history of user-item interactions (adding an item to personal bookmarks, buying an item from e-commerce sites, counting consumption times on content sites and so on).

Generally, explicit feedback is more precise, but harder to collect. Recommender systems that try to optimize for explicit ratings generally perform worse than ones working with implicit feedback [7].

Fantlab (fantlab.ru) is an electronic library similar to Goodreads (goodreads.com), providing information about books and authors, allowing users to rate books and write reviews.

In this project we try to build a recommender system for Fantlab. We build two models:

1. A content session-based model that tries to predict the next work that a user will read using his interaction history so far;

2. A purely collaborative filtration model using `LightFM` library [1].

Finally, we compare the models' performance.

1.1 Team

Yury Popov: Collecting data, datasets creation, Collaborative filtration, Content-based model.
Andrey Nepomnyashchy: Collecting data, datasets creation, embeddings creation, simple recommendations, fine-tuning the models.

2 Related Work

There are plenty of collaborative filtering algorithms using implicit, explicit or both types of feedback.

2.1 Collaborative models

A collaborative recommender system trains on user-item preference matrix R, whose rows are users $u \in U$, columns are items $i \in I$, and $r_{u,i}$ is a measure of interaction between u and i. It is clear that generally R is a rather sparse matrix. The current industry standard for collaborative recommender system are factorization methods, which try to approximate R by a product of (low-rank) matrices P_U (user features) and Q_I (item features). The recommendations themselves are then done by choosing items i with highest predicted value $p_u \cdot q_i$ for each user u.

In case of prevailing explicit feedback, R contains aggregated ratings that users left for items, and the algorithm of choice is usually Alternating Least Squares (ALS) [3], which tries to minimize the Euclidean distance between the R and $P_U Q_I$. ALS is implemented in various Python libraries, such as `implicit`. However, it is known that RMSE may not be the best loss function to express user preferences [7] (particularly, it does not preserve relative rankings of any given user). Hence, in the recent years, various attempts to construct models for implicit feedback were made.

When dealing with implicit feedback, the elements of R are usually binary, indicating whether a user u interacted with an element i (particularly, elements such that user u interacted with are called *positive* for u, and all others *negative*).

At first glance, by binarizing R we lose valuable data by ignoring interaction degrees (that is, there is no obvious measure to tell the model how much a user liked/disliked an item). However, one can justify this approach by noting that the data in interaction matrices is likely not missing at random, that is, absence of interaction generally means implicit dislike. Moreover, for various content providers there is much more implicit feedback than explicit one (users do not tend to leave ratings, however, it is easy to track their activity). In this setting, the goal is to rank all positive elements above negative ones, and suitable loss functions are Bayesian Personalized Ranking (BPR) [4] and Weighted Approximate-Rank Pairwise (WARP)[12] loss. Both BPR and WARP losses are implemented in `LightFM` Python library, the latter often subjectively claimed to achieve better results.

One of the common problems of collaborative recommender systems is the problem of *cold start*: because such models are trained on interactions data, they struggle to learn useful representation of users and items with few previous interactions. One of the approaches to deal with this problem involves including item- and user-metadata in the features, allowing the model to learn representations of previously unknown elements. `LightFM` allows one to provide binary features for users and items (essentially, tags). A more sophisticated approach was recently realized in DropoutNet [10] paper, where authors used neural feature embeddings and dropout to explicitly address the cold start problem. In [11] the authors built a recommender system for explicit feedback on an anime tracking site adding graph embeddings to DropoutNet architecture.

We must note that Fantlab has its own purely collaborative statistic-based model recommendation system ("Invisible connections" and "Recommended" section) which seems similar to BM25[5]. There are "invisible connections" between users which are generally correlation coefficients of different types. Individual recommendations are built by taking the mean of the preference weights of user's like-minders.

2.2 Content-based models

Purely content-based recommender systems are generally employed at e-commerce sites, where the task at hand is to recommend a next item to a user based on their actions during the current session. Such recommender systems are called *session-based*. One of the interesting techniques in session-based recommender system is called Prod2Vec, which aims to learn item representations by their co-occurrences in sessions, similarly to Word2Vec. Prod2Vec approach was studied and improved extensively in the recent years, see, for example, [9, 6]. Having the item embeddings, the logical approach is to train an RNN on user transactions (see, for example [8] and references therein).

Large content-streaming services like Okko employ hybrid recommendation systems allowing for both explicit and implicit feedback (in Russian).

3 Model Description

We decided to approach the solution from two sides, that is, we build purely collaborative and purely content-based models. For the collaborative approach we use the `LightFM` factorization machine and for the content approach we apply a recurrent network (using `torch.nn.GRU`) which accepts ids of works which user interacted with and tries to predict the next interacted work.

Below are some Fantlab-specific findings that we had to take into account in the process of building datasets and training the models:

1. Fantlab is not a very popular site, which leads to lack of data (number of items and number of users are comparable, as opposed to the classical recommender system setup, where $n_{users} \gg n_{items}$);
2. Almost all feedback on fantlab comes in form of explicit user ratings (although there are some ways for a user to implicitly interact with an item, such as adding a book to a bookshelf, or writing a novel, it occurs extremely rarely). So one should expect user-item preference matrices to be rather sparse (think how many books you read per year);
3. Each item comes with user-provided metadata, such as tags and descriptions;
4. Sparsity of user features (generally user profiles contain minimal information, if any).

3.1 Collaborative model

Since Fantlab user-content interaction system has almost no implicit feedback, we are limited to using explicit feedback only. Therefore, we have two approaches to construct the collaborative model:

1. Work with explicit ratings, trying to predict ratings for yet unseen items for each user;
2. Work with implicit feedback, trying to rank all observed interactions above unseen ones for each user.

Two factors were taken into account while choosing between the approaches:

- Initial data analysis showed that ratings on Fantlab are highly right-skewed, with average user mark being around 7.5 on a scale from 1 to 10 (that is, users almost always "like" the items on the site, and ratings themselves are not very representative). Hence, we deem the property of predicting the fact of the interaction more important than the property of predicting an exact rating;
- Despite ratings being not very good expressions of how much a user liked a certain work, their *relative* order for a given user is a more or less objective

criterion for comparing two works. Models working with explicit feedback tend to ignore relative orderings (there is no obvious way to tell RMSE, the loss function for these models, to preserve them), essentially destroying this important data.

Thus, we choose to follow the second approach. We choose WARP loss as the function to optimize during training. As discussed above, models working with implicit feedback deal with binary interaction matrices. A simple solution to adapt our situation to this setting would be to binarize the ratings, but doing so, we would deprive our model of valuable information (it is clear that ratings 1 and 10 do not mean the same thing). Luckily, `LightFM` allows for a workaround to incorporate the rating data in the model: interaction weights. We gave a weight to each interaction that takes into account the average mark of the user and the date of the interaction. For instance, it seems reasonable that newer marks should have higher weights than older ones. The general algorithm of time-weighting the interactions is as follows:

1. min-max normalize mark dates (user-by-user or globally);
2. map normalized timestamps to $[\epsilon_{time}, 1]$ (the oldest interactions thus get weight ϵ_{time});
3. optionally apply some nonlinear transformation to the weights (we use power transforms)

A more detailed discussion on the choice of our approach, WARP loss, and interaction weights can be found in the notebook `collaborative.ipynb`.

Remark: It appears that an "ideal" model for our task would be able to compute WARP loss on an interaction matrix with arbitrary integer elements (in this way we could treat low ratings as explicit negative feedback). However, we have not found any public realization of such model (the developers of Okko's recommender system apparently wrote such model themselves, see here (in Russian)).

3.2 Session-based model

Our session-based model a simple recurrent network (with some linear layers on top that process each output hidden state independently) similar to ones used in language modelling tasks. For each user, it accepts precomputed embeddings (see below) of work descriptions that user read in chronological order and predicts the next item in the sequence. To augment the data and enhance the training speed, we broke down long sequences into subsequences of length ≤ 20 (a measurable share of users have ≥ 1000 work marks!). The sequences of length < 20 are padded from the left (right padding would give a wrong representation of users' history if one wants to predict future interactions). The loss in this case is a simple cross-entropy with number of classes equal to number of items observed in the train dataset. By now, this model does not accept any user features, which can be seen both as a disadvantage (knowing users' metadata would probably improve prediction quality) and as an advantage (this model can give meaningful predictions for users it did not observe in the train set). We cannot see an obvious way for this model to take explicit marks into account.

4 Dataset

The data to train the models was obtained by Fantlab public API. Particularly, we downloaded the work infos for all works with ≥ 50 marks (`work_infos.json.gz`), containing such information as work descriptions and tags (provided by site users), and all marks (until 2022-08-05) for such works (`work_marks_csv.gz`). Totally, we have 127 tags, 23867 works, 61739 users and 7368410 marks. The code to download the data can be found in the project files (see notebook `dataset_creation.ipynb`).

4.1 Collaborative model

`LightFM` allows to provide additional features for users and items during training, which may increase the ability of the model to deal with the cold start problem. While user data on the site is rather scarce, works are usually provided with user-generated tags, which were parsed and used to train the model (see file `item_features.json.gz`).

4.2 Session-based model

For the content session-based model we obtained different embeddings of work descriptions:

- Pretrained embeddings by `rubert-tiny` model;
- Pretrained GloVe embeddings from `natasha/navec`;
- Embeddings obtained by `gensim/Doc2Vec`;
- Embeddings obtained by `gensim/Word2Vec`;
- TF-IDF embeddings.

Sentence embeddings by `Word2Vec` and `natasha` were obtained by averaging the individual word embeddings.

Pre-embedding, work descriptions were stripped of stopwords, normalized by `pymorphy2`'s `MorphAnalyzer`, and tokenized except for `rubert-tiny` embedding, because the tokenizer of the model works using byte-pair encoding, which allows one to use inflected word forms. The embeddings can be found in `data/interim` project folder.

5 Experiments

5.1 Metrics

To evaluate the predictions for a single user, we use the precision@k metric:

$$p@k = \frac{relevant\,predicted}{k},$$

where k is the number of works with the highest predicted rank ($k = 20$ in this project). In other words, we simply count the proportion of recommended items in the top-k set that a user interacted with. We have chosen the $p@k$ because it represents well the quality of the model and easy to implement. A global metric for the whole dataset is obtained by averaging $p@k$ for each user and is denoted by $p@k$ as well.

5.2 Experiment Setup

The train-test split was done in the following way: we split interactions by time, so the recommender model trains on users' interactions in the train period and predicts the interactions for the same users in the test period. Particularly, our train period lasts from 2012-01-01 to 2020-07-01 (0.8 time quantile of the marks' timestamps) and test period lasts from 2020-07-01 to 2022-05-08 for both models (to get comparable results). Note that the train period is rather long: in order to have a model with a sensible predictive ability, one must have enough data to train on, but as explained above, interactions on Fantlab are not very frequent.

Before creating the dataset, the interactions are filtered: we drop users with less than 10 marks in the train period (users with few marks are not likely to get good predictions) and items with less than 10 marks in the test set (works with few marks are not likely no get recommended). Moreover, the metric was computed only on users that have interactions in the test set. The code for constructing the datasets can be found in `src/preprocessing/datasets.py`.

Remark: The final model metric is rather sensitive to the date of train-test split. For instance, with smaller train set, the number of users in it is lower, but the users are generally more active and thus have a higher chance of interacting with predicted works in the test set, which is itself larger, so one should expect a higher metric value (surely there are other patterns). We assume that in industrial recommender systems the actual train-test proportion is found out empirically (probably using A/B tests with external metrics).

5.2.1 Collaborative model

We train `LightFM` model using sparse matrices of user-work explicit interactions and their weights.

The model and training hyperparameters are:

- dimensions of user and item features: 30;

- regularization terms: `item_alpha = 1e-5, user_alpha=1e-5`;
- learning rate: `lr = 0.05`;
- epochs: 5.

Remark: In order to reproduce the experiment results, use `n_threads = 1`, although the model is sufficiently stable, so results should not change much from run to run with `n_threads > 1`.

The experiements with the collaborative model can be found in the notebook `collaborative.ipynb` in the project files.

5.2.2 Session-based model

Additionally to the train-test time split, for session-based model we made a separate validation set by splitting 20% of users in the train set.

Recurrent model was trained on sequences of `rubert-tiny` embeddings (of dimension 312) of work descriptions for each user. The structure of the recurrent model is as follows:

- A GRU with two hidden layers with hidden size 256 and cell dropout = 0.5;
- two linear layers of size 500;
- ReLU activation between linear layers;
- dropout between linear layers: 0.5

We train the recurrent model with the following settings:

- epochs: 30;
- batch size: 256;
- learning rate: `lr = 0.01`;
- `Adam` optimizer with weigh decay: `wd = 1e-6`;
- `ReduceLROnPlateu` scheduler with `patience = 0`;

We have slightly sacrificed model stability in favor of the training speed and the results. Regularizing the model (increasing weight decay, adding batch norm) will improve its stability but will strongly decrease training speed at the same time. The experiments with this model can be found in the notebook `recurrent_recommender.ipynb`.

5.3 Baselines

We tried to use different vector representations of the descriptions with simple similarities functions (like k nearest neighbors, cosine similarity) to predict user preferences. Navec and RuBert-tiny embeddings could not even identify works within one cycle or universe (for example, Sapkovsky's "Witcher" or "Hyperion" by Dan Simmons), which is perhaps not that surprising, because the relationship between similar words in model-based embeddings is not assumed to be linear(?).

We built a simple "recommeneder system" that aggregates user preferences vector by summing TF-IDF embeddings of their observed work descriptions with interaction weights discussed above and tries to find most similar items by measuring cosine similarity (see class `LinearRecommeder` in the project files). It achieved test $p@K \approx 0.01$. An interesting fact is that Doc2Vec was good enough in finding works of the same genre and stylistics, and we can assume that its embeddings are worthy too.

6 Results

`LightFM` model with interaction weights and no item features achieved $p@k \approx 0.0695$ on train and $p@k \approx 0.3386$ on test set. Surprisingly, adding item features (in form of tags) offered negligible or no metric improvement, while significantly increasing the training time (apparently, this is a frequent situation) Particularly, adding item features to our final model, we achieve $p@k = 0.3744$ on the train dataset and $p@k = 0.0689$ on the test set. Hence, adding tags overfits the model. On the other hand, adding interaction weights to the model improves $p@k$ by around half percent.

Recurrent recommender also shows decent results and gives stable $p@k \approx 0.052$ score. Despite this result is more than one percent worse than collaborative model, content model is more flexible for modification and fine-tuning and will probably be able to compete with `LightFM` with more data. Unfortunately, RNN fits 5 times longer, almost half an hour without dataset creation.

Model	Train	Test	Time
LightFM without features	**0.3386**	**0.0695**	**4m 10s**
LightFM with features	0.3744	0.0689	8m 30s
Recurrent content	–	0.0521	29m 19s

Table 1: Results of our models.

7 Conclusion

In this project, we proposed and built recommender systems for `fantlab.ru`. Simple approach to use item embeddings and interaction weights to build personal recommendations was not successful, nevertheless TF-IDF empirically works decent enough to fill the “Similar books” section.

`LightFM` collaborative filtration turned out to be the most efficient model working significantly faster than the content model which showed lower score than `LightFM`. However, recurrent model has the right to exist, being well-modifiable.

8 Future work

Some of the things that would be interesting to try in the future:

- Check the distribution of recommended works for both models. It is known that recommender systems suffer from the problem of recommending the same popular items to all users. One way to deal with that is to re-weight the interactions according to their popularity;
- Try to come up with meaningful user features for both models;
- We saw that work descriptions embeddings alone are enough to construct a working recommender system. One can try to construct a two-stage recommender system that uses work embeddings as features for its content part;
- Find a working representation of tag features that improves the quality of the model (one of the approaches would be to remove some tag groups that are useless to the model);
- Another idea on how to “make tags to work” in our task is to make use of the natural graph-like work structure on the site (works can share tags, authors, time of writing, etc) and learn graph features of the works;
- Apart from statistical learning, existing Fantlab recommender system uses various rules to exclude undesired items from recommendations (for example, books by authors that user disliked in the past are not recommended). We expect that introduction of such rules can improve our models.

References

[1] Kula, M. (2015). Metadata Embeddings for User and Item Cold-start Recommendations. In T. Bogers & M. Koolen (Eds.), Proceedings of the 2nd Workshop on New Trends on Content-Based Recommender Systems co-located with 9th ACM Conference on Recommender Systems (RecSys 2015) 2015. (Vol. 1448, pp. 14–21)

[2] M. Grbovic, V. Radosavljevic, N. Djuric, N. Bhamidipati, J. Savla, V. Bhagwan, and D. Sharp. E-commerce in your inbox: Product recommendations at scale. In Proceedings of the 21th ACM SIGKDD International Conference

on Knowledge Discovery and Data Mining, KDD '15, pages 1809–1818, New York, NY, USA, 2015. ACM.

[3] Hu Y., Koren Y., Volinsky C., Collaborative Filtering for Implicit Feedback Datasets, 2008 Eighth IEEE International Conference on Data Mining, 2008, 263-272, DOI: 10.1109/ICDM.2008.22.

[4] Rendle S., Freudenthaler C., Ganther Z., Schmidt-Thieme L., BPR: Bayesian personalized ranking from implicit feedback, UAI '09: Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, 2009, 452–461

[5] Robertson S., Zaragoza H., The Probabilistic Relevance Framework: BM25 and Beyond, Foundations and Trends in Information Retrieval, 2009 (3), 4, 333–389, DOI: 10.1561/1500000019

[6] Srilakshmi M., Chowdhury G., Sarkar S., Two-stage system using item features for next-item recommendation, Intelligent Systems with Applications 2022 (14), 200070, DOI: 10.1016/j.iswa.2022.200070

[7] Steck H., Training and testing of recommender systems on data missing not at random, Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, DOI: 10.1145/1835804.1835895

[8] Quadrana M., Karatzoglou A., Hidasi B., Cremonesi P., Personalizing Session-based Recommendations with Hierarchical Recurrent Neural Networks, arXiv: 1706.04148

[9] Vasile F., Smirnova E., Conneau A. Meta-Prod2Vec - Product Embeddings Using Side-Information for Recommendation, arXiv:1607.07326

[10] Volkovs M., Yu G., Poutanen T., DropoutNet: Addressing Cold Start in Recommender Systems, Advances in Neural Information Processing System, 2017 (30)

[11] Sun, M.M., Vu, T., Yang, R, Wang, A. (2021). DeepAniNet : Deep NLP-based Representations for a Generalizable Anime Recommender System

[12] Weston J., Bengio S., Usunier N., Wsabie: Scaling Up To Large Vocabulary Image Annotation, Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence.