
A HYBRID APPROACH FOR NEWS RECOMMENDER SYSTEM USING OPTIMIZATION METHODS

A PREPRINT

✉ **Alexander Smirnov**

Department of Information and Analytical Systems
Saint Petersburg State University
Russia, Saint Petersburg
ru.alexander.smirnov@gmail.com

✉ **Elena Mikhailova**

Department of Information and Analytical Systems
Saint Petersburg State University
Russia, Saint Petersburg
e.mikhaylova@spbu.ru

March 11, 2021

ABSTRACT

Recommender system is an essential part of any social media application. Most recommender systems now use a hybrid approach, combining collaborative filtering, content-based filtering, and other approaches. Most common problems in the field of hybrid recommenders are cold start and data sparsity [Çano, 2017]. In this paper we address the abovementioned problems by proposing a hybrid weighted news recommender system which combines different approaches.

Keywords hybrid recommender systems · content-based recommender · collaborative recommender · optimizations

1 Introduction

Recommender system is a crucial part of every application that operates with content and user activity. Enormous amount of information leads to the problem that user is not able to find relevant content.

Recommender systems have been studied to present items, such as movies, music, and books [Duan et al., 2011] [Min and Zhu, 2013] [He et al., 2012]. The term now has a broader connotation, describing any system that produces individualized recommendations as output or has the effect of guiding the user in a personalized way to interesting or useful objects in a large space of possible options. Such systems have an obvious appeal in an environment where the amount of on-line information vastly outstrips any individuals capability to survey it.

Recommender systems are now an integral part of some e-commerce sites such as Amazon.com and CDNow [Schafer et al., 1999]. It is the criteria of individualized and interesting and useful that separate the recommender system from information retrieval systems or search engines [Belkin and Croft, 1992]. The semantics of a search engine are matching: the system is supposed to return all those items that match the query ranked by degree of match. Techniques such as relevance feedback enable a search engine to refine its representation of the users query, and represent a Simple form of recommendation.

There are three main types of recommendation methods: memorybased, model-based, and hybrid [Bobadilla et al., 2013]. Memory-based methods [Delgado and Ishii, 1999] usually use similarity metrics to obtain the distance between two users or two items. Model-based methods use demographic, content, or aggregated information to create a model that generates recommendations. Hybrid [Claypool et al., 1999] methods combine different types of recommenders to gain better performance.

Common approaches, such as collaborative filtering, have their own problems: cold start, scalability and data sparsity. Content-based approaches suffer from the fact that we have to somehow represent recommended item in feature space.

So we are going to present a hybrid recommender system.

To be consistent during the paper we list some domain specific vocabulary with their meanings:

- Rating: expression or preference
 - explicit (direct from user, e.g. user rated film)
 - implicit (inferred from user activity, e.g. user stopped watching movie after 5 minutes)
- Prediction: estimate of preference
- Recommendation: selected items for user
- Content: attributes, text, etc; everything about item

The remainder of this paper is organized as follows:

- Section 2 describes the relevant related work
- Section 3 describes input data
- Section 4 explains our modular design and architecture
- Section 5 provides tests and experiments validating our systems results
- Section 6 explains future work
- Section 7 presents conclusions

2 Related work

According to the study [Dacrema et al., 2019], deep learning techniques are not supposed to beat conceptually and computationally simpler algorithms, so we won't touch them.

Our goal is to choose optimal algorithm for each of the following tasks:

- **Collaborative filtering:** generating predictions about the interests of a user by collecting preferences or taste information from other users. It is based on the assumption that if a person A has the same opinion as a person B on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person; there were many studies on this topic, but paper [Rendle et al., 2019] proves that well-tuned basic SVD++ approach beats newly presented algorithms
- **Content-based filtering:** is based on the assumption that people who liked items with certain attributes in the past, will like the same kind of items in the future as well. It makes use of item features to compare the item with user profiles and provide recommendations
- **Session filtering:** is selecting candidates based on user's activity on current session
- **Popularity filtering:** uses information such as number of views, shows, comments, etc.
- **Demographic filtering:** uses demographic data such as age, gender, education, etc. for identifying categories of users
- **Time-based filtering:** ranking news is a way that more recent items have higher scores

Also there is several ways [Burke, 2002] to combine recommenders between each other:

- **Weighted:** The scores (or votes) of several recommendation techniques are combined together to produce a single recommendation
- **Switching:** The system switches between recommendation techniques depending on the current situation
- **Mixed:** Recommendations from several different recommenders are presented at the same time
- **Feature combination:** Features from different recommendation data sources are thrown together into a single recommendation algorithm
- **Cascade:** Features from different recommendation data sources are thrown together into a single recommendation algorithm
- **Feature augmentation:** Output from one technique is used as an input feature to another
- **Meta-level:** The model learned by one recommender is used as input to another

We face the problem that millions of news are theoretically suitable for being recommended, so it is not correct to use abovementioned methods on such large corpus of data. Instead, as stated in [Covington et al., 2016] paper, we want to implement candidate generation → ranking pipeline to reduce number of candidates.

3 Input data

As we solving domain specific task, we have domain specific data. Our data consist of information about news, so it includes following tables:

metadata

The metadata about news. Here we have an *item_id* column, which stands for unique item identifier, afterwards *date*, that shows when this item was released. The *source_id* column stands for id of a publisher of this news item. *category* column means category of current news item. This category is taken from news text by keywords.

<i>item_id</i>	<i>date</i>	<i>source_id</i>	<i>category</i>
1	2021-01-08 22:08:39	9	politics, conflicts
2	2021-01-09 10:28:58	5	IT, social media
3	2021-01-09 14:20:34	12	accident
⋮	⋮	⋮	⋮

Table 1: news metadata

content

Content table hold news texts in a *news_content* column. This is the primary information that is being used by our recommender system, because we are able to extract a lot of valuable data from text, such as topics.

<i>item_id</i>	<i>news title</i>	<i>news content</i>
1	Azerbaijan denies reports on construction of Turkish air bases in the country	Information that Turkey will create air bases ...
2	Durov announced the massive transition of WhatsApp users to Telegram	Telegram developer Pavel Durov said in his channel ...
3	Passenger plane that disappeared from radar crashed	Passenger plane taking off from Jakarta, disappeared ...
⋮	⋮	⋮

Table 2: news item

shows & views

As we operate with user activity we should have user logs, such as what items were clicked at what time, so we have separate table that contains this information. *shows* stands for *item_id* was shown to the *user_id*, *views* is if *item_id* was clicked by the *user_id*.

<i>user_id</i>	<i>item_id</i>
10	1
10	2
23	1
23	3
23	2
38	3
38	1
⋮	⋮

Table 3: news shows

<i>user_id</i>	<i>item_id</i>
10	1
10	2
23	1
38	3
⋮	⋮

Table 4: news views

emotions & comments

We have an explicit feedback that user may leave on a news if he wants to. These options includes leaving emoji, which is one of {😊, 😐, 😞, 😡, ❤️} or comment, which we can analyze afterwards.

<i>user_id</i>	<i>item_id</i>	<i>emotion_id</i>
10	1	1
10	2	3
23	1	3
38	3	2
\vdots	\vdots	\vdots

Table 5: users' emotions

<i>user_id</i>	<i>item_id</i>	<i>comment</i>
10	1	that's great
10	1	wish it will continue
23	2	whatsapp is not competetive anymore
\vdots	\vdots	\vdots

Table 6: users' comments

users' subscriptions

Each piece of news is being posted by some feed and user have an ability to subscribe to the feed. So it may give us useful information if user prefers content from one feed to content to another feed. So following table illustrates if *user_id* subscribed to the *source_id*.

<i>user_id</i>	<i>source_id</i>
10	9
23	5
\vdots	\vdots

Table 7: users' subscriptions

4 Overview of our approach

Our goal is to combine state-of-the-art approaches in recommender systems.

Solution consists of 2 parts:

- **Candidate generation:** lowering number of items to recommend. These candidates are intended to be generally relevant to the user with high precision. The candidate generation part only provides broad personalization
- **Ranking:** applying state-of-the-art algorithms to rank candidates generated on previous step

Architecture provided below on fig. 1:

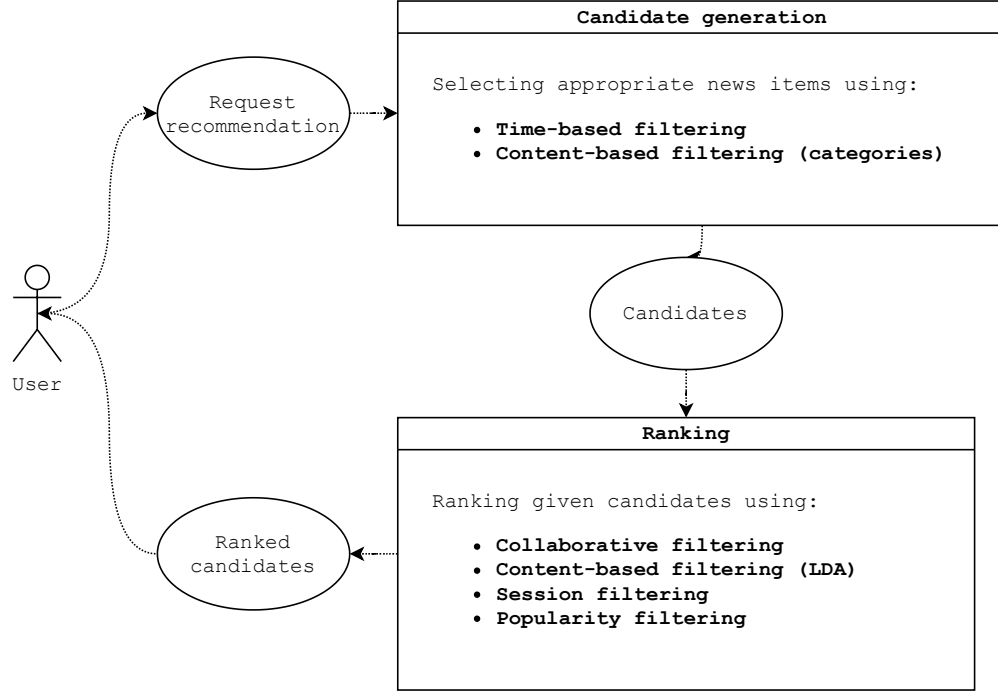


Figure 1: architecture

Candidate generation stage not only filters items, but also attaches weights to them.

4.1 Candidate generation components overview

Goal of candidate generation step is to remove generally irrelevant items so further algorithms won't suffer from the amount of data. Both time-based filtering and content-based filtering have their own weight at start. As user more interacting with the system, content-based filtering increasing its weight.

Using recommenders described below, we attach score to each item and pick top n (e.g. 50000) items.

4.1.1 Time-based filtering

As we operating with news data, first filter is the time filter. This part consists of 2 steps:

- **Filtering:** remove all items which are older than 3 days
- **Ranking:** attach weights to all items left from filtering

For ranking we will use following formula:

$$r_i = \frac{(v_i - 1)}{(t - t_i + 2)^G}$$

- r_i – score for $item_i$
- v_i – number of views of $item_i$
- t – time right now, t_i – time of creation of $item_i$, $t - t_i$ – hours passed since item created
- G – gravity factor

The score decreases as $t - t_i$ increases, meaning that older items will get lower and lower scores. v_i gets subtracted by -1 to negate submitter's view. $t - t_i$ increased by 2 so even if $t - t_i = 0$ gravity factor G will take effect.

Below are show dependencies between score and hours since creation:

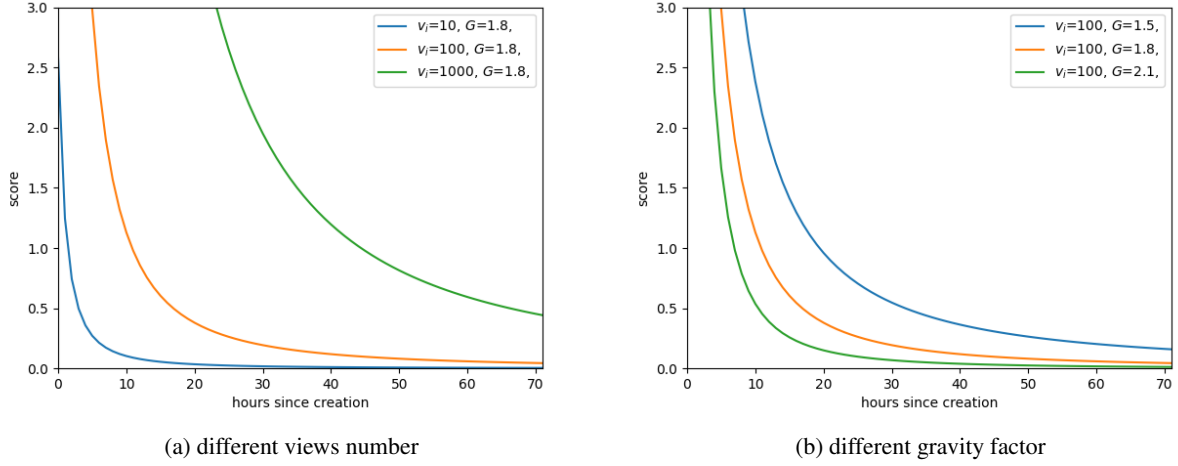


Figure 2: time-based ranking scoring

At the end scores are normalized by min-max normalization.

4.1.2 Content-based filtering (categories)

We are representing each news item as a vector of confidence values (how strong each item belongs to category).

item_id	politics	IT	social_media	...	confilcts
1	0.55	0	0	...	0.3
2	0	0.81	0.62	...	0
⋮	⋮	⋮	⋮	⋮	⋮

Table 8: text vectors

Tagging is made in the following way: we have a dictionary of words that belongs to categories:

word	category
trump	politics
crash	accident
telegeram	IT
⋮	⋮

Table 9: words' categories

Confidence is taken from word's tf-idf metric alongside all corpus.

As user interacts with the system, we are forming his preferences vector in the following way:

$$\text{user_vector} += \text{action_score} * \text{item_vector} \quad (1)$$

Where action score is taken from table:

<i>action</i>	<i>score</i>
shown but now viewed	-1
viewed	0.5
emoji or comment	0.5
read till the end	1

Table 10: actions' scores

As we now have both user's and items' vectors, we are able to find similarities between them:

$$r_i = 1 - \cos(\text{user_vector}, \text{item_vector}) \quad (2)$$

As was told before, content-based filtering have its own impact weight, which is small at start (we don't want to restrict user from content just because he made some random clicks), but it inscreases as user interacts with the system and we may make predictions about his categorial preferences.

4.2 Ranking components overview

4.2.1 Collaborative filtering

We are using SVD++ algorithm.

For collaborative filtering recommendation we should have something known as user-item matrix which may be formed from user activity from tables (cite tables).

We use the modification of Funk MF, which factorized the user-item rating matrix as the product of two lower dimensional matrices, the first one has a row for each user, while the second has a column for each item. The row or column associated to a specific user or item is referred to as latent factors.

$$r_{ui} = \sum_{f=0}^n H_{u,f} W_{f,i}$$

While Funk MF is able to provide very good recommendation quality, its ability to use only explicit numerical ratings as user-items interactions constitutes a limitation. Modern day recommender systems should exploit all available interactions both explicit (e.g. numerical ratings) and implicit (e.g. likes, purchases, skipped, bookmarked). To this end SVD++ was designed to take into account implicit interactions as well. Compared to Funk MF, SVD++ takes also into account user and item bias. The predicted rating user u will give to item i is computed as:

$$r_{ui} = \mu + b_i + b_u + \sum_{f=0}^n H_{u,f} W_{f,i}$$

4.2.2 Popularity filtering

For measuring news item popularity following data can be aggregated: **shows**, **views**, **emotions**, **comments**.

<i>item_id</i>	<i>shows_num</i>	<i>views_num</i>	<i>emotions_num</i>	<i>comments_num</i>
1	1043	231	52	7
2	828	478	78	11
3	163	25	5	0
\vdots	\vdots	\vdots	\vdots	\vdots

Table 11: aggregated popularity data

So when we apply information about news item popularity, we are able to give them scores via following algorithm:

min-max normalizing each of *shows_num*, *views_num*, *emotions_num*, *comments_num*, then dividing by 4 (to have 1 as max after sum), and sum all of these values.

4.2.3 Content-based filtering

latent Dirichlet allocation (LDA)

4.2.4 Session filtering

same as previous

5 Evaluation

For evaluation we have information about what recommendation list was given to each user, and what is the source of the recommendation:

<i>user_id</i>	<i>recommendation_list</i>	<i>content_based_filtering</i>	<i>collaborative_filtering</i>
2	{(2, 0.91), (1, 0.74), (3, 0.23)}	{(2, 0.45), (1, 0.54), (3, 0.08)}	{(2, 0.92), (1, 0.4), (3, 0.3)}
1	{(3, 0.73), (1, 0.69), (2, 0.15)}	{(2, 0.6), (1, 0.44), (3, 0.04)}	{(2, 0.58), (1, 0.58), (3, 0.14)}
\vdots	\vdots	\vdots	\vdots

Table 12: recommendations' logs

- *recommendation_list*: list of recommendations that consists of pairs (*item_id*, *score*)
- *content_based_filtering* & *collaborative_filtering*: sources of recommendation

As we use weighted sum of our recommenders, we have unique boost values for every user:

<i>user_id</i>	<i>content_based_filtering</i>	<i>collaborative_filtering</i>
1	0.25	1
2	1	0.5
\vdots	\vdots	\vdots

Table 13: boost values

To illustrate score calculation of recommendations, take a look at 1st row of table **recommendations' logs**. We see recommendation $r = \{(2, 0.91), (1, 0.74), (3, 0.23)\}$ for user $u = 2$, so we should find boost values for this particular user in table **boost values**: content-based filtering boost b_{cbf} for u is 1, collaborative filtering boost b_{cf} is 0.5. So final score is calculated in the following way: $0.91 = 0.45 * b_{cbf} + 0.92 * b_{cf} = 0.45 * 1 + 0.92 * 0.5 = 0.45 + 0.46$.

Also we have information about **shows** and **views**, so we are able to track what item in recommendation list was clicked and what item was skipped, so depending on this information we can track if user liked item or not.

6 Further research

7 Summary

Results show that combining different approaches leads to rise of users' involvement.

References

Erion Çano. Hybrid recommender systems: A systematic literature review. *Intelligent Data Analysis*, 21:1487–1524, 11 2017. doi:10.3233/IDA-163209.

- L. Duan, W. N. Street, and E. Xu. Healthcare information systems: data mining methods in the creation of a clinical recommender system. *Enterprise Information Systems*, 5(2):169–181, 2011. doi:10.1080/17517575.2010.541287. URL <https://doi.org/10.1080/17517575.2010.541287>.
- Fan Min and William Zhu. Mining top-k granular association rules for recommendation. *Proceedings of the 2013 Joint IFSA World Congress and NAFIPS Annual Meeting, IFSA/NAFIPS 2013*, 05 2013. doi:10.1109/IFSA-NAFIPS.2013.6608601.
- Xu He, Fan Min, and William Zhu. A comparative study of discretization approaches for granular association rule mining. *Canadian Conference on Electrical and Computer Engineering*, 37, 12 2012. doi:10.1109/CCECE.2013.6567823.
- J. Ben Schafer, Joseph Konstan, and John Riedl. Recommender systems in e-commerce. In *Proceedings of the 1st ACM Conference on Electronic Commerce, EC '99*, page 158166, New York, NY, USA, 1999. Association for Computing Machinery. ISBN 1581131763. doi:10.1145/336992.337035. URL <https://doi.org/10.1145/336992.337035>.
- Nicholas Belkin and W. Croft. Information filtering and information retrieval: Two sides of the same coin? *Commun. ACM*, 35:29–38, 12 1992. doi:10.1145/138859.138861.
- J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, 2013. ISSN 0950-7051. doi:<https://doi.org/10.1016/j.knosys.2013.03.012>. URL <https://www.sciencedirect.com/science/article/pii/S0950705113001044>.
- Joaquin Delgado and Naohiro Ishii. Memory-based weighted-majority prediction for recommender systems. 01 1999.
- M. Claypool, Anuja Gokhale, Tim Miranda, Paul Murnikov, Dmitry Netes, and M. Sartin. Combining content-based and collaborative filters in an online newspaper. In *SIGIR 1999*, 1999.
- Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM Conference on Recommender Systems, RecSys '19*, page 101109, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362436. doi:10.1145/3298689.3347058.
- Steffen Rendle, Li Zhang, and Yehuda Koren. On the difficulty of evaluating baselines: A study on recommender systems. 05 2019.
- Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12, 11 2002. doi:10.1023/A:1021240730564.
- Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, New York, NY, USA, 2016.