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
ITMO university
Russia, Saint Petersburg
e.mikhaylova@spbu.ru

May 27, 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.

Field of news recommendation has its own specific: news are getting old really fast and they don't need to be recommended.

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 contains overview of our approch
- Section 4 describes input data
- Section 5 explains our modular design and architecture
- Section 6 provides tests and experiments validating our systems results
- 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 newely 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 cadidates 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` $\rightarrow$ `ranking` pipeline to reduce number of candidates.

3 Overview of our approach

Out 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:

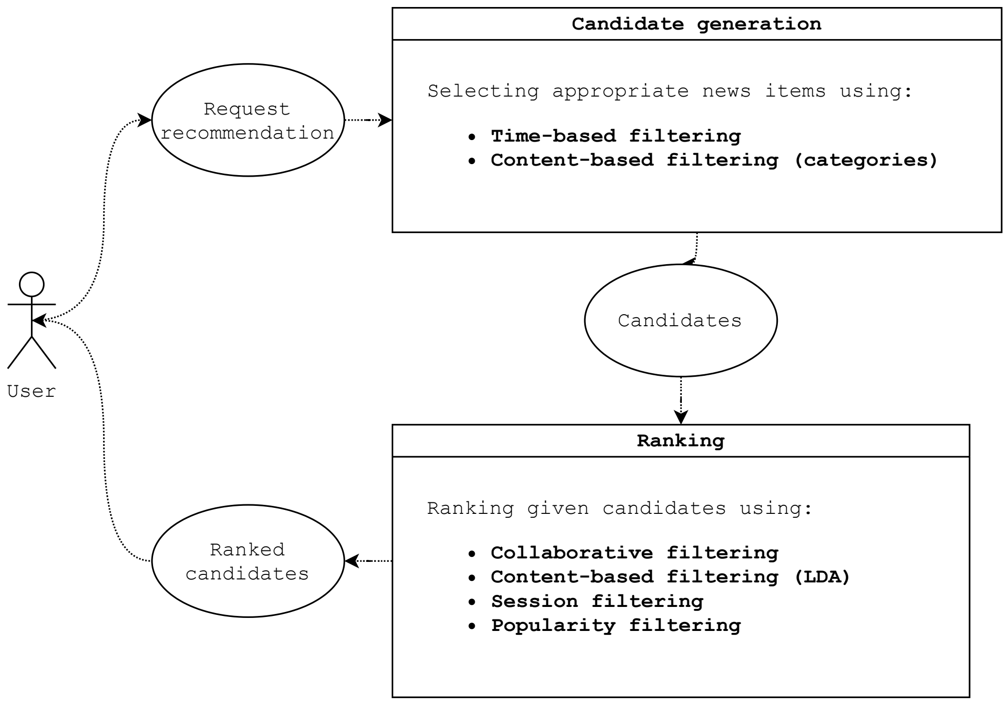


Figure 1: architecture

4 Input data

As we solving domain specific task, we have domain specific data. Out 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 indentifier, 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.

item_id	*date*	*source_id*	*category*
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.

item_id	*news title*	*news content*
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 clilcked 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*.

user_id	*item_id*
10	1
10	2
23	1
23	3
23	2
38	3
38	1
⋮	⋮

Table 3: news shows

user_id	*item_id*
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.

user_id	*item_id*	*emotion_id*
10	1	1
10	2	3
23	1	3
38	3	2
⋮	⋮	⋮

Table 5: users' emotions

user_id	*item_id*	*comment*
10	1	that's great
10	1	wish it will continue
23	2	whatsapp is not competetive anymore
⋮	⋮	⋮

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*.

user_id	*source_id*
10	9
23	5
⋮	⋮

Table 7: users' subscriptions

5 Our approach

Lets describe in detail candidate generation ranking pipeline.

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

5.1 Candidate generation components overview

Goal of candidate generation step is to remove generally irrelevant items so furher 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.

5.1.1 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:

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

- 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 substracted 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. By default we set $G = 1.8$ and we increase it if we want to prioritize newer news over old and vice versa.

Below are show dependencies between score and hours since creation:

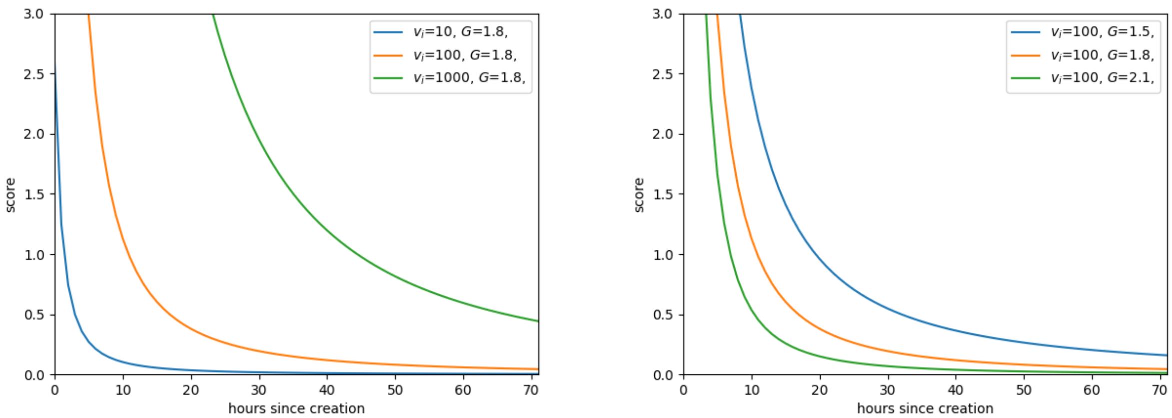
At the end scores are normalized by `min-max` normalization.

5.1.2 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



(a) different views number

(b) different gravity factor

Figure 2: time-based ranking scoring

word	*category*
trump	politics
crash	accident
telegeram	IT
⋮	⋮

Table 9: words' categories

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

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

There is limited number of words, but during the day new words are being added and every night item vectors are being recalculated.

We have actions while user interacts with the system and weights for items for each interaction (table 10).

action	*score*
shown	-1
viewed	2
emoji or comment	1
read till the end	1

Table 10: actions' values

All data is aggregated and each news item have an action score depending on what user had done with it. For instance, suppose user had viewed and item and left an emoji. Total score will be: -1 (user was shown an item) $+2$ (user viewed an item) $+1$ (user left an emoji) $= 2$.

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

$$u_i = \sum_{j=1}^{n} v_j \times s_j \tag{2}$$

- u_i – vector of $user_i$

- v_j – vector of $item_j$
- s_j – action value of $item_j$

User vector has same dimension as item vector.

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

$$r_{ui} = \cos(\theta) = \frac{\mathrm{u} \cdot \mathrm{v}}{\|\mathrm{u}\|\|\mathrm{v}\|} = \frac{\sum_{i=1}^{n} u_i v_i}{\sqrt{\sum_{i=1}^{n} u_i^2}\sqrt{\sum_{i=1}^{n} v_i^2}} \tag{3}$$

- r_{ui} – rating from $user$ to $item$
- u – vector of $user$
- v – vector of $item$

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.

5.2 Ranking components overview

5.2.1 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.

$$r_{ui} = \sum_{f=0}^{n} H_{u,f} W_{f,i} \tag{4}$$

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:

$$r_{ui} = \mu + b_i + b_u + \sum_{f=0}^{n} H_{u,f} W_{f,i} \tag{5}$$

5.2.2 Popularity filtering

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

item_id	*shows_num*	*views_num*	*emotions_num*	*comments_num*
1	1043	231	52	7
2	828	478	78	11
3	163	25	5	0
⋮	⋮	⋮	⋮	⋮

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.

$$r_i = \frac{shows\_num\_norm}{4} + \frac{views\_num\_norm}{4} + \frac{emotions\_num\_norm}{4} + \frac{comments\_num\_norm}{4} \tag{6}$$

5.2.3 Content-based filtering (LDA)

For Content-based filtering we should somehow vectorize news items and represent user preferences via these vectorized news. We will use Latent Dirichlet Allocation (LDA), which is a generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. For example, if observations are words collected into documents, it posits that each document is a mixture of a small number of topics and that each word's presence is attributable to one of the document's topics.

So we are able vectorize text by the measure of how each text belongs to each category, from 0 to 1. For example, suppose we have 3 topic, that were extracted by LDA model:

topic	*word*	*score*
1	dollar	0.3
1	bank	0.2
1	money	0.15
2	sugar	0.4
2	cooking	0.3
3	IT	0.25
3	hacker	0.2

Table 12: words' topics

And we have following text: "Hackers use mobile emulators to steal millions of dollars". Its vectorized form is going to be $[0.3, 0, 0.2]$.

All calculations are done as was told in 5.1.2.

5.2.4 Session filtering

We want to instantly react on user's actions, so we applying session filtering in the following way: trying to find similar item to those, user have just watched. So

$$r_i = \sum_{k=0}^{n} similarity\{current\_item\_vector, last\_viewed\_vector_k\} \times weight_k \tag{7}$$

where $weight_k$ is the weight of last viewed vector. Weight is bigger if item was seen more recently.

5.3 Combining

So having all of this information allows us to tune the impact weight of each recommender using grid search. So we are optimizing weight that are making impact at each recommender.

The metric we are optimizing is MAP@20 that stands for Mean Average Precision.

6 Evaluation

6.1 Overview

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

- *recommendation_list*: list of recommendations that consists of pairs (*item_id*, *score*)

user_id	*recommendation_list*	*content_based_filtering*	*collaborative_filtering*
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)}
⋮	⋮	⋮	⋮

Table 13: recommendations' logs

- *content_based_filtering & collaborative_filtering*: sources of recommendation

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

user_id	*content_based_filtering*	*collaborative_filtering*
1	0.25	1
2	1	0.5
⋮	⋮	⋮

Table 14: 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. For instance, let's consider situation that user clicked on 4th recommendation from recommendation list. That means that first 3 recommedations are bad, and we should penalty recommenders that prioritized these items, considering the fact that 1st item is the worst:

$$penalty = \frac{1}{num\_of\_item\_in\_rank} \tag{8}$$

6.2 Metrics

We are using three metrics to measure the quality of our recommenders:

- RMSE
- NDCG
- MAP@20

7 Summary

Recommendation system has been widely used in different areas. Collaborative filtering focuses on rating, ignoring the features of items itself. In order to better evaluate customer preference on news, we use LDA model to calculate customer preference on news topics.

In order to forecast rating on news, we take similarity of customers and correlation between customers and news into consideration. Experiment shows that our hybrid recommendation method based on features performances better in our social media app.

What we contribute is we proposed a new hybrid recommendation method based on features to improve the performance.

Results show that combining different approaches leads to rise of users' involvement. Firstly we had only a collaborative recommender and user averagely spend on news tab around 2 minutes. Now when we applied hybrid recommender users spend in average 5 minutes on news tab.

We use the average method to set the weight to adjust the predicted rating in this article, whose rationality needs to be further improved. Moreover, our method has yet to be tested on other data sets for its performance.

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.