Recommendation of YouTube Videos

M. Brbić, E. Rožić and I. Podnar Žarko
Faculty of Electrical Engineering and Computing (FER), Zagreb, Croatia
firstname.lastname@fer.hr

Abstract - YouTube is a huge video-sharing service with hundreds of millions of users and hundreds of thousands of videos being uploaded every day. Thus, recommendation of YouTube videos to a single user is a challenging problem which cannot be solved by simply reusing the prevailing recommendation methods.

The paper presents a specific recommendation algorithm for YouTube which relies on the data retrieved through the YouTube Data API. A cloud-based application integrates the proposed algorithm and offers a web interface to end users. The paper presents a preliminary analysis of the recommendation quality and lists YouTube Data API limitations which influence the design of recommender systems for YouTube videos.

I. INTRODUCTION

Recommender systems and algorithms stem from fields such as cognitive science, approximation theory, information retrieval and forecasting theories. In the mid-1990s recommender systems have become an independent research area and today we can classify recommender algorithms in three basic categories: content-based recommendations, collaborative recommendations and hybrid approaches [1].

Content based recommendations predict future user actions and preferences based on their past actions. Collaborative recommendations try to find similarities between a user and other users who have already used the system, and based on past actions of similar users recommend some content which might be interesting to the user. Hybrid systems combine both approaches to offer better recommendation performance.

Recommendation algorithms depend widely on the available data set which captures past user behavior, similarities between the data and also on context such as time of year or user's current location. Collaborative recommendation systems typically produce their recommendations based on user and item information while taking into account user context. Thus, the two-dimensional User×Item matrix which shows which items have been consumed by all system users in the past is extended to multidimensional space to take context information into consideration. The easiest way to solve the problem related to context and avoid extending the matrix to multidimensional space is reduction-based recommendation proposed in [2]. Reduction based recommendation uses only ratings that correspond to the specified context and reduces greatly the input dataset.

YouTube recommendation is a specific problem which cannot be solved by simply reusing existing recommendation methods. There are about 485,000,000 videos in repository [4] and that number is growing every minute. Approximately, 48 hours of video are uploaded every minute, resulting in nearly 8 years of content uploaded every day [5]. A large number of users and great dynamicity of YouTube content represent special challenges to recommender systems. Moreover, YouTube gives limited access to user and video information through the YouTube Data API which represent another major difficulty. For example, it is not possible to access a viewing history for a user but it is possible to find out which videos a user has added to his/her favorites or uploaded over time and also which videos a user has rated positively (within the last 60 days [6]. By reusing such data, a User×Item matrix would be almost a zero matrix because the probability that two users have added the same video to their favorites is very small. Therefore, the standard collaborative recommendation methods are not applicable. The problem with content based approaches is that almost every user has a set of videos (his favorites and likes) which is too small, and if a recommender systems would use such small amount of information regarding a user to find the right video from the set of millions of videos in the YouTube repository, the resulting recommendation quality would be extremely poor. Therefore, we propose a different approach which is specific for YouTube recommendation.

In this paper we describe our algorithm and initial experimental evaluation of our algorithm. The paper is organized as follows: In Section 2 we present our algorithm and the idea behind it. Section 3 describes the implementation of the algorithm and related technologies. Section 4 preliminary evaluation results of the recommender system while Section 5 closes the paper with conclusions and lists possible system improvements.

II. ALGORITHM DESCRIPTION

Since we have concluded that a content-based approach is simply inapplicable for YouTube recommendation in any form because of the lack of usable information about a user, we have decided to find another criterion to identify similar users for a target user U for whom the recommendation is made. Of course, the limitation is the aforementioned set of data that we have access to via the YouTube API.

We can get a list of user's friends, but recommendation of videos that his/her friends have liked in the past is probably not a good solution for many reasons. The user does not have to share the same taste as his friends, and it is likely that the user will share videos with his friend who have the same taste as he/she (YouTube has an option

share) while a large number of users do not have many friends on YouTube. Thus, as users that are most similar to user U (his related users), we have decided to choose those users that have uploaded videos which user U has rated positively or added to favorites. The assumption is that if user U has liked a video which the user R has uploaded in such a degree that he added that video to favorites or rated it positively, the user U will probably also like other videos which user R uploads, rates positively or adds to his favorites. This assumption is similar to the follower effect which is characteristic for Twitter, where user U is the follower of user R. However, we have realized that this assumption could be valid only for those videos that are in the same category as the video which user U likes. For example, if user U liked a music video which user R uploaded, it is likely that user U and user R have the same taste in music. However, we cannot make any assumption about the interests of user U in other categories like sports or entertainment compared to user R.

The first step of the algorithm finds all videos that user U has liked or added to favorites in the past, and then it identifies the users that have uploaded those videos and the categories of those videos. Those are the related users of user U. This information is stored for every user and is updated at each subsequent rerun of the algorithm (login to the system or request for a recommendation). The information is stored per each user as a Related_user×Category matrix which gives the number of videos from a given category which user U has liked or added to favorites from a related user. Since the matrix will most probably be very scarce (it seems very unlikely that one user will be related to another in more than one or two categories) it is most appropriate, in our opinion, to use a data structure like a map (dictionary) to store it.

The second step of the algorithm finds all the videos that the related users of a user U have liked, added to favorites or uploaded in the same category which relates the two users. Those videos are considered to be of interest to user U. It is obvious that the recommendations will be better for more active users i.e. users with greater number of related users. The assumptions are that the users who upload videos are active YouTube users, as it is often the case with such users, and that user U has videos that he/she added to favorites or has positively rated.

Finally, after obtaining videos of all related users which represent potential recommendations, we have to rank them. For that we have decided to use the information about the number of views of each video (*viewCount*) and the number of times a video had been positively (*likeCount*) or negatively (*dislikeCount*) evaluated. That information represents, in a way, the recommendation of the whole YouTube community. We also took into account two parameters that are specific for our algorithm. Those are the *appearanceNumber* and *userFactor*. All parameters used for ranking of the videos are summarized in table 1.

For the purpose of ranking we take the logarithm of the number of views of a video to get an order of magnitude because the number of views of a video spans from several tens to several hundreds of millions and we wanted it to have a relatively equal part in the ranking process as the other parameters. The overall assessment of a video we decided to punish according to the proportion of negative evaluations of the video. We found empirically that it is desirable for better results to attenuate the impact of that portion and therefore we decided to take the square root. If that is not done then videos with very few views and even fewer ratings, all of which are positive, are ranked very high. That is not really good because there are a lot of such videos which are liked by close friends and acquaintances, but few others would find them interesting.

If more than one related users have added the same video in their favorites or liked it, extra weight is given to the video in the ranking process. This is represented by the *appearanceNumber* parameter. For example, if a video appears in the potential recommendations of two related users then the *appearanceNumber* of that video is set to 2. However, a situation in which the *appearanceNumber* is greater than 1 will be very rarely satisfied because the probability that two or more related users in a limited period of time have added to favorites or positively rated the same video (among all the videos on YouTube) is very small. The *userFactor* is the information that is held by the Related_user×Category matrix, i.e. it is the number of times user U liked or favorited a video from a related user in a specific category. Since the *userFactor* is less significant than the *appearanceNumber* and is much more likely to be greater than 1 we attenuated the value of *userFactor* by using the square root.

TABLE 1. – RANKING PARAMETERS

Parameter name	Parameter description
viewCount	number of views of the YouTube video
likeCount	number of positive ratings (likes) of the YouTube video
dislikeCount	number of negative ratings (dislikes) of the YouTube video
userFactor	the number of times the user liked or favorited a video in the same category as this video that the related user, in whos activity this video was found, uploaded
appearanceNumber	the number of times this exact video is recommended in the current recommendation (usualy one)

The final formula used for ranking videos identified as potentially interesting to a user is the following:

$$\text{rating} = \frac{\log(\text{viewCount})}{\sqrt{\frac{\text{dislikeCount}}{\text{dislikeCount}+\text{likeCount}}}} \cdot \sqrt{\text{userFactor}} \cdot \text{appearance Number} \quad (1)$$

where rating is a score given to each video.

In cases when the *dislikeCount* is equal to zero it is necessary to make a correction of its value. We found empirically that in this case it is desirable to set the value

of the *dislikeCount* to one. If a video is really good and has a large number of positive ratings and a large number of views then adding a one dislike to such video will not have a major impact on its overall rating. If a video has a small number of positive ratings and a small number of views this will greatly reduce its rating, but this effect is positive because we usually don't want to recommend such a video (obviously this video is interesting to a small group of people, and therefore it is not likely that it will be interesting to user U).

III. IMPLEMENTATION

The recommender system is implemented as a web-application[1] written in the Java programming language on the Google App Engine cloud computing platform [7] using YouTube Data API libraries for extracting required data sets from YouTube.

The user interface enables the user to enter his/her YouTube username and choose among two options – make a new recommendation or view past recommendations.

When a user makes the first request for a recommendation, the application contacts YouTube and gets all the favorite videos of that user and all the videos that he liked (rated positively) as far in the past as the YouTube API will let it or keeps track itself (60 days). The information about the users who uploaded those videos and the categories of those videos are extracted and stored in the Google App Engine datastore as RelatedUser entities which represent the Related_user×Category matrix for that user. Also, a User entity is stored containing the username and the date and time of the last recommendation request for that user. Every subsequent time that same user makes a recommendation request the application retrieves from YouTube only the activity (liked and favorite videos) from the last time the user made a recommendation request and updates the Related_user×Category matrix for that user, i.e. updates some and adds new RelatedUser entities if necessary.

After all the related users and corresponding categories for a user are found the application contacts YouTube again and retrieves all the videos the related users have liked, added to favorites or uploaded (in the right category) since the last recommendation request of the user using the application (or 14 days if first use). First the *appearanceNumber* and *userFactor* parameters for each video are set and then the rating for each video is calculated by the algorithm formula. The videos are then sorted by the calculated rating in a descending order and the first 25 are taken as the recommendation.

Before the recommended videos are shown to the user a Recommendation entity is made with two parameters: time and date of its making and the time span it took for recommendation. Also, for each of the recommended videos a RecommendedVideo entity is made which are all put as children of the just made Recommendation entity and which hold information about the videos that were recommended. That is needed for users to be able the see their past recommendation and more importantly for the application to be able to note if the recommended video has been clicked on by the user it was recommended to.

If the user chooses on the main page to view his past recommendations the application simply gathers information about the past Recommendation entities of that user and presents them to the user in the form of the time the recommendation was made and the time span it was made on. If a user clicks on one of them the past recommendation is shown in its exact original form. The application simply does that by finding the relevant RecommendedVideo entities in the datastore.

Whenever a user of the application clicks on a recommended video the link does not take him directly to the URL of the video. Instead it takes him to a separate URL on the application on which operates a servlet that takes the information from the GET request and based on them retrieves the relevant RecommendedVideo entity from the datastore. It gets the real URL of the video from the retrieved RecommendedVideo entity but before it redirects the user to that URL it marks the entity (or rather the video that the entity represents) as clicked and puts it back in the datastore. In that way the usage of the application can be easily tracked and some simple tests of quality of the recommendations can be made.

IV. RESULTS

In order to view the results objectively the reader should know that the collected data for the evaluation of the algorithm was collected from very similar people in very similar circumstances and should also be aware of the fact that we made a small improvement on the algorithm at the time of testing which we believe improves the quality of the recommendations greatly. At first the maximum amount of time (60 days) of recent activity of related users was used for the first recommendation. We realized that because of that the first recommendation will have a great chance of having high ranked videos that the user had already watched and that it will divert the user from using the application again. So we decided to reduce the time span for that first recommendation to 14 days, a time period we considered long enough to generate sufficient videos and yet short enough for those videos to be recent. Because of that the data collected after the change will be presented separately.

The application was used by a total of 113 users. Of those 113 users 29 have no related users and were thus useless for the algorithm. There are two reasons why a user would have no related users. The first is the obvious one – the user has no favorites and has not rated any video in the 60 days prior to using the application and the second is that the user's YouTube profile is private. Those 29 users are not taken into further account; all calculations are made on the remaining 84.

The average number of related users over all users is 15.86. It is interesting to observe the average number of related users of only the users who have made more than one recommendation and those who have made only the first and never came back. The average number of related users of the users that made only one recommendation is 14.84 while the average number of related users of the

[1] http://givemeclips.appspot.com

users that made 2 or more recommendations is 27.92. This is a strong indication that users with more related users get better recommendations.

Of the 84 valid users 7 had very few (1 to 3) and apparently very inactive related users because no videos were recommended to them. If we disregard those 7 users we are left with 77 users of which 13 have returned after making the first recommendation, which is 16.88%. If we now separate those 77 users by the time span of their first recommendation (60 and 14 days i.e. before and after the change) we get a group of 52 and a group of 25 users respectively. Of those 52 users that made the first recommendation before the change to the application 7 returned after the first recommendation, which is 13.46%. Of the other 25 users 6 returned, which makes 24%. This clearly indicates that the change to the application/algorithm was a positive one.

Since the first recommendations are a bit different than the others because they cover a longer time period and often recommend a lot more videos, at least in the small dataset we have gathered, we decided to analyze them separately.

The average number of videos per recommendation for all recommendations is 18.10 and the average number of visited videos per recommendation is 1.69. For only the first recommendations that were made on a 60 day time span those averages are 22.96 and 1.60. For only the first recommendations that were made on a 14 day time span those averages are 17.36 and 1.36, which is expected. The most interesting results are the averages for the recommendations that were not the first. They have a lot lower average of videos per recommendation, only 9.44, but also a quite higher average of visited videos per recommendation - 2.18. This could be interpreted as an indicator that, although later recommendations have fewer videos those videos are more interesting. That is consistent with the idea behind this algorithm which is to recommend videos that the user would normally run into very hard because they are not very popular yet or got lost in the sea of videos, but that are interesting to the user because of his/hers specific interests.

The collected data is also used for evaluation of the ranking formula (1). The most visited video is the first video i.e. the best ranked video. The first video of a recommendation was selected 33 times in 104 recommendations and out of 175 videos selected in total. Top 5 videos were selected a total of 83 times, and top 10 videos a total of 118 times. That means that 47.4% of all selected recommended videos were ranked among top 5 and 67.4% among top 10 videos in a recommendation. These results indicate that better ranked videos are in fact more interesting to the user and that the use of formula (1) is justified.

V. DISCUSSION AND CONCLUSION

In this paper a recommendation algorithm for YouTube was presented which relies on the data retrieved through the YouTube Data API. This algorithm finds all YouTube users who uploaded videos which the target user has liked or added to favorites and categories of these videos and then stores that information in a Related_user×Category matrix. The algorithm then finds the videos which those related users had recently uploaded, liked or added to favorites and that are of the corresponding category and ranks them using formula (1) to form a recommendation.

Our recommendation algorithm depends a lot on the activity of the related users or the user itself. If a user is not very active on YouTube and has a few related users, the recommendation will most probably not work very well for him/her. This problem is a consequence of the limited information that we can obtain using YouTube Data API.

A definite specificity of the algorithm is the amount of the recommended videos. Since they are generated from the recent activity of the related users there will be only a small amount of them on a day-to-day basis, unless, of course, a user has a huge amount of related users. But from the testing that we managed to do in this short period of time the amount of related users ranges from just a few to a maximum of 45. The problem lies in the fact that most people just watch YouTube videos. A very large amount of them do not even have user accounts, and a lot of others use them very rarely and for specific purposes (music playlists, etc.). Average YouTube users rarely rate (click LIKE) on a video they like. That is obvious from simply observing the number of view counts on YouTube videos in contrast to the number of ratings. On the other side, for users that do have accounts and do rate videos occasionally, when they really like something, this recommendation algorithms makes a really personally tailored recommendation made of videos that will not show up that easily on their standard YouTube recommendation. Thus our algorithm can be used in cooperation with a more standard recommendation like e.g. the YouTube official recommendation which can recommend extra videos when our algorithm cannot find a sufficient number of videos[2].

There is a possibility to expand the algorithm so that it always generates enough recommended videos. This could be done by making the algorithm recursive in a manner that the related users of the related users of a user can be found and their activity checked. In our opinion this would be unscalable network-wise because all that activity generates traffic to and from YouTube. But if the algorithm would be run on YouTube servers directly that could be a possible solution.

Of course, that would then solve the biggest setback of the whole problem of recommending YouTube videos and that is the limitations of the information that can be retrieved from YouTube. The biggest limitation of all for this algorithm in its current form is the inability to catch events like negative ratings. For example, a user rates a video positively and because of it gets a related user. The user then gets a recommendation based on that related user and rates a video negatively. The related user that was responsible for that negative rating should be removed from the related users list. But that information is not retrievable. One possible, indirect way to solve this problem is to remove from the related users list those

[2] The YouTube official recommendation for a user is retrievable by YouTube Data API.

related users that are responsible for recommended videos that the user clicks on very rarely or never in the recommendations presented to him.

Due to all the limitations we must conclude that recommending YouTube videos in this kind of manner can hardly be a substitution for the official YouTube recommendation since it lacks information and speed because of the way the information that can be retrieved is retrieved. But we believe that this is a good new way to look at recommendation problems in services with similar characteristics as YouTube, which are very dynamic, and we argue that our idea of finding related users and generating a user specific reduced User×Item matrix has potential which would be worth exploring by implementing it on a real system with immediate access to all relevant information.

REFERENCES

[1] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," IEEE Transactions on Knowledge and Data Engineering, Vol. 17, No. 6, pp. 734–749, June 2005.

[2] G. Adomavicius, R. Sankaranarayanan, S. Sen, and A. Tuzhilin, "Incorporating contextual information in recommender systems using a multidimensional approach," ACM Trans. Information Systems, Vol. 23, No. 1, Jan. 2005.

[3] M. Deshpande, G. Karypis, "Item-based top-N recommendation algorithms", ACM Transactions on Information Systems, Vol. 22, No. 1, pp. 143-177, Jan. 2004.

[4] YouTube Search Results for *, http://www.youtube.com/results?search_query=*&search=Search

[5] YouTube Statistics, http://www.youtube.com/t/press_statistics

[6] YouTube APIs and Tools, http://code.google.com/intl/hr-HR/apis/youtube/getting_started.html#data_api

[7] What is Google App Engine?, http://code.google.com/appengine/docs/whatisgoogleappengine.html