3 Large Scale Video Recommendations

3.1. Introduction

The largest online shopping sites today offer millions of items for sale in its catalog. By the same way, video portals like YouTube also have dozens of millions of items, with thousands of new content being published every day. Since choosing among so many options is challenging for consumers, recommender systems have emerged in response to this problem, and are now a fundamental component of video services like Netflix, YouTube, DailyMotion, etc. The goal of a recommender system in this case is to recommend items that are likely to fit the user expectations. Today, recommender systems are deployed on hundreds of different sites, serving millions of consumers. As discussed previously, one of the earliest and most successful recommender technologies is collaborative filtering [11, 12, 13, 14]. Collaborative filtering (CF) works by building a database of preferences for items by users, and has been very successful in both research and practice. However, there remain important research questions in overcoming two fundamental challenges for collaborative filtering recommender systems.

The first challenge is to improve the scalability of the memory-based collaborative filtering algorithms. These algorithms are able to search tens of thousands of potential neighbors in real-time, but the demands of modern Internet systems are to search tens of millions of potential neighbors. Further, existing algorithms have performance problems with individual users for whom the site has large amounts of information. For instance, if a site is using browsing patterns as indications of item preference, it may have thousands of data points for its most valuable customers. These “long customer rows” slow down the number of neighbors that can be searched per second, further reducing scalability. The second challenge is to improve the quality of the recommendations for the consumers. Users need recommendations they can trust to help them find items they will like. In some ways these two challenges are in conflict, since the less time an algorithm spends searching for neighbors, the more scalable it will be, and the worse its quality. For

PUC-Rio - Certificação Digital Nº 1122198/CA

this reason, it is important to treat the two challenges simultaneously so the solutions discovered are both useful and practical.

3.2. Item-Item Video Recommendations

As discussed previously, the idea behind an item-item recommendation engine is, for any given item, finding a set of items that is most similar to the item in question. Similarity is measured using a combination of input data, generally structured in a bi-dimensional matrix with the first dimension representing users and the second items. The ultimate goals of an item-item recommendation system are to predict how users would rate an item that is not yet rated, and to recommend items from the collection. In a video recommendation system, an item-item collaborative filtering could be used to provide recommendations in an approach *"who liked this video might also like these ..."*.

With respect to user ratings, recommendation engines may use different types of feedback. Ideally, explicit feedback is preferred, with users explicitly indicating their preferences. Netflix uses explicit user feedback through star ratings as shown in Figure 6, for example, while YouTube uses *"thumbs up"* and *"thumbs down"* as its explicit feedback, as shown in Figure 7.

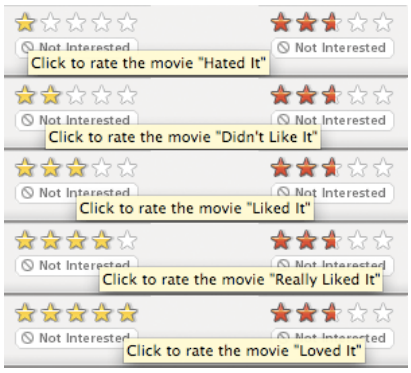

Figure 6 - Netflix Star Ratings for explicit feedback

PUC-Rio - Certificação Digital Nº 1122198/CA

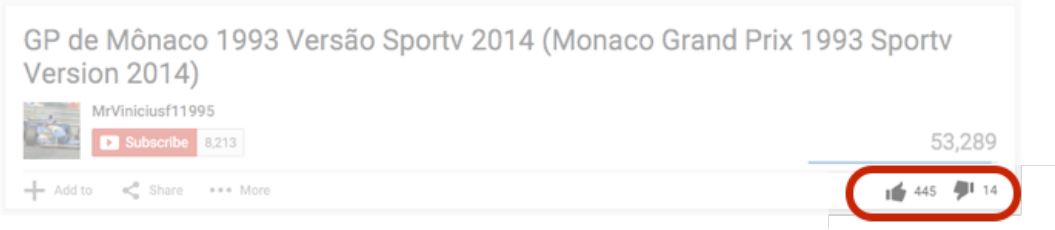

Figure 7 - YouTube *"Thumbs up" / "Thumbs Down"* explicit feedback

However, there are situations where such information is not available. In these cases, it is necessary to infer user preferences through implicit feedback [55], which indirectly represents a user's preferences based on his/her behavior. Implicit feedback is obtained from user navigation history, the list of products bought, and even from the mouse path on specific screens. For videos, this implicit feedback could be obtained through the playback duration, video seek (rewind or forward), player visibility during the playback and different user interactions with the player beyond others [56].

After obtaining user feedback, for example, information that a user start playing a particular video, the input can be stored in a square matrix representing users and items, where, in this case, each element denotes whether an item was accessed by a user. Thus, considering each item separately, we have a multidimensional vector with each dimension representing a user. Consequently, as the number of distinct users in the system increases, the number of dimensions in the vector also increases. Typically, large video portals have tens or even hundreds of millions of unique visitors per month, which means that their item vectors will have a similar number of dimensions.

Therefore, the problem of obtaining the similarity between two specific items can be translated into one of obtaining the similarity between the two multidimensional vectors representing the items. As shown in previous chapter, one of the possible approaches for doing this is by calculating the cosine between these vectors [8], and is the approach that will be considered for the scope of this thesis.

Let us suppose that two items, say I_1 and I_2 in $\mathbb{R}^M$, where M is the number of users. This work adopted the cosine function to compute the similarity between these two items. Such similarity is than calculated as follow:

PUC-Rio - Certificação Digital Nº 1122198/CA

$$Similarity(I_1, I_2) = \cos\big(\measuredangle(I_1, I_2)\big) = \frac{\langle I_1, I_2 \rangle}{\|I_1\| . \|I_2\|},$$

where $\measuredangle$, $\langle \ .,. \rangle$ and $\|.\|$ represents, respectively, the angle between the two vectors, the usual inner product in $\mathbb{R}^M$ and the Euclidian vector norm.

Having calculated the similarity between a specific item and all others, one can then obtain the items that are most similar to one another, and thus, create an item-item recommendation that takes into account the feedback from users. However, it is important to remember that for each piece of feedback received, for example, for each accessed item, the similarity between this and the other items can change, and hence the similarity between the current item and all the others in the vector must be recalculated.

In this scenario, considering an environment with millions of users and millions of items it would be necessary, for each piece of feedback received, to recalculate the cosine between two vectors with millions of dimensions to ensure the similarity graph is updated in real-time. This is necessary because with each feedback, one of the vectors' coordinate changes, which in turn changes the cosine value and the similarity between this vector and all other item vectors.

As an illustration consider a video website with *N* different videos (items), and *M* unique viewers (users). This information can be represented by a $M \times N$ matrix *F* in which items are represented as columns and users as rows. This matrix can be used to track feedback, i.e., users who have demonstrated an interest in the items. Whenever there is a new entry, the matrix must be updated and the similarity graph recalculated. This recalculation is necessary because when a matrix value changes, one of dimensions of the item vector also changes, so the cosine between this vector and the others may be different. This difference changes the similarity between two items, and, consequently, can modify the similarity ranking between one item against the others. Figure 8 depicts a concrete case where user U_3 plays video I_3. In this case element F_{U_3,I_3} of the feedback matrix must be updated, and all item pairs $I_3 : I_1$, $I_3 : I_2$, ..., $I_3 : I_n$ must have their similarity recalculated as shown. For a model with 4 million items, this means 3,999,999 similarity calculations every time an item is viewed.

PUC-Rio - Certificação Digital Nº 1122198/CA

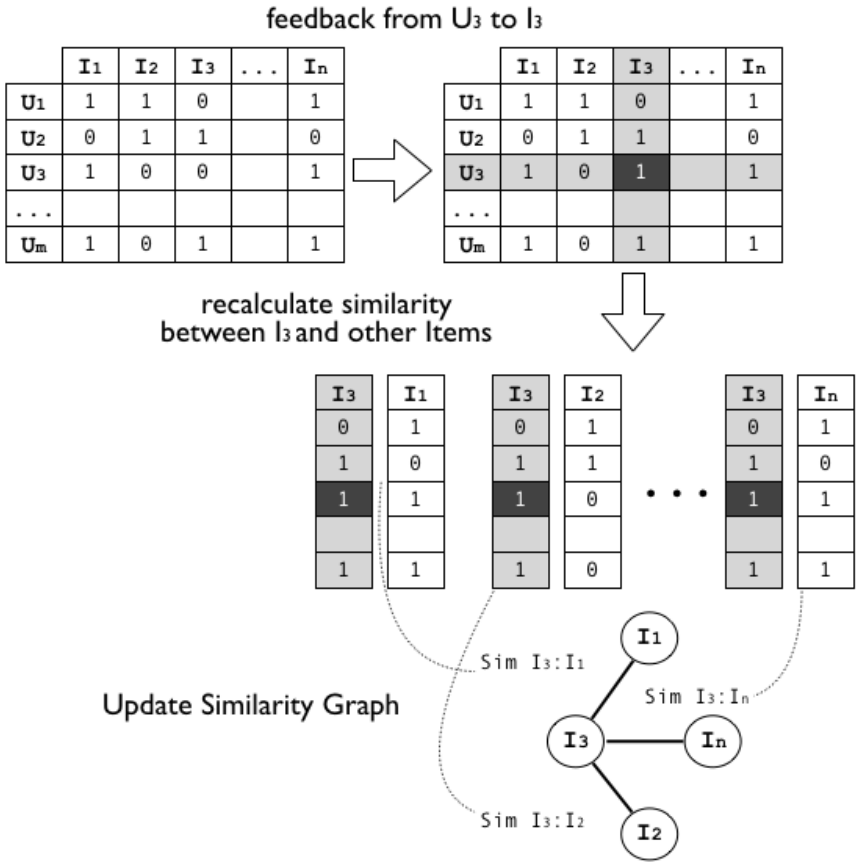

Figure 8 - Similarity recalculation process for a single piece of feedback

When using binary feedback, it is possible to avoid the entire recalculation only by checking if the same dimension of the other vectors has been already set. If it is not set, the recalculation is not necessary, since the cosine will not change. Consequently, the average complexity is much better than the worst case complexity for binary feedbacks. However, for a large number of users and items, perform a real-time similarity calculation in a practical scenario still being a challenge in terms of computing power.

Another challenge associated with item-item recommendation is the variation in item relevance over time, that is, the introduction of temporal dynamics [37]. In the real world, the perception and popularity of an item is constantly changing as new items are introduced. Similarly, user preferences evolve. Thus, the system must take into account temporal effects in mapping the dynamic and variable nature of the interactions between users and items.

PUC-Rio - Certificação Digital Nº 1122198/CA

This is even more critical when the interest over items is very volatile. To illustrate such volatility, Figure 9 below shows the number of video views of 5 different news videos from the most important and popular news program in Brazil, the *Jornal Nacional*, from TV Globo. All videos are from same program edition, and were published in the same day and around the same hour, during the broadcast of the program on TV.

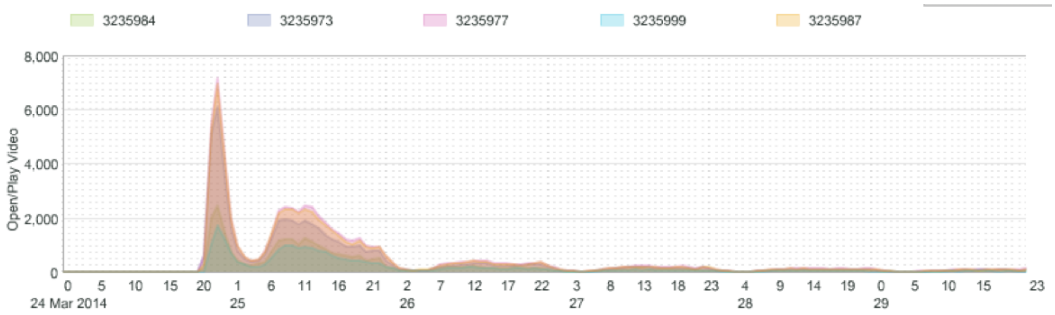

Figure 9 - Video views from 5 news videos from the *Jornal Nacional*

PUC-Rio - Certificação Digital Nº 1122198/CA

As expected, there is a spike in audience in the first hours after publication, however, the content almost become irrelevant to users after few days, since past news might have very low interest for the most part of viewers. So, in most cases, recommend news from days, months or years in the past could not be interesting for the users.

To tackle this part problem in practice, it is possible to associate a lifetime with each feedback, item, and user, so that the values of each dimension of the vector representing the item can be adjusted according to the temporal variation. Thus, a positive feedback could become neutral over time, given that this feedback does not necessarily represent current user behavior, or even given that this item is no longer relevant in the context.

Considering all aspects, a theoretical model to obtain item-item recommendations could be implemented by structuring the users and items in a two-dimensional array containing the feedback, and by calculating the cosine between vectors of items to obtain the similarity graph between items. Moreover, considering the temporal dynamics, which in the case of content providers, especially breaking news, is essential, we would need a third index in this array to define a life span for each piece of feedback or group thereof.

Again, the major challenge to implement such theoretical model for large scale applications, such as major video services as YouTube, is how to manipulate very large datasets. This becomes even more complex when the speed of similarity graph updating is business critical, for example, in breaking news portals. In those cases, the manipulation of large datasets must be done in near real-time, since only the usage o temporal dynamics could be enough the provide fresh and relevant news recommendations.

Since this real-time computation is not critical for YouTube service, they implemented a batch-oriented pre-computation approach rather than online calculation of recommendations. This has the advantages of allowing the recommendation generation stage access to large amounts of data with ample amounts of CPU resources while at the same time allowing the serving of the pre-generated recommendations to be extremely low latency. The most significant downside of this approach is the delay between generating and serving a particular recommendation data set [57].

Netflix also relies on offline computation to generate its video recommendations [58]. However, as [58] clearly states, offline results can easily grow stale between updates because the most recent data is not incorporated. In order to consider the most recent data, Netflix implemented a combination of offline computation with real-time data streams, also using cloud computing platforms. However, there are no details about how this hybrid architecture was implemented and what information is considered in the online computation, and, most important, how it was designed to scale to millions of users and thousands of items.

Furthermore, Netflix content tends to be much less volatile then breaking news content, being much less sensitive to temporal dynamics. In this scenario, the online computation is used much more as a refinement of the offline models, which could not be enough for more dynamic environments. Another important reminder is that despite the large number of active users, and, consequently volume of feedback data, Netflix has a much more limited catalog of items when compared to YouTube, for example, which definitely reduces overall complexity of collaborative filtering, for example.

This thesis focus is a scenario that is similar to YouTube and Netflix, however, with characteristics that aren’t present in neither services. Very large portals such

PUC-Rio - Certificação Digital Nº 1122198/CA

Globo.com, CNN, BBC, among others usually have very large number of users such YouTube and Netflix, and catalogs with millions of objects, such YouTube, however, their temporal dynamics needs are much more relevant. Furthermore, the real-time computation is strategic for services dealing with breaking news, which is not the case of YouTube and Netflix.

So, this thesis proposes an architecture that could address these services needs, enabling real-time computation to generate fresh recommendations to large portals through a collaborative filtering approach. The next chapter describes the implementation details of such architecture and how cloud computing was used to enable a large scale processing.

PUC-Rio - Certificação Digital Nº 1122198/CA