Recommendation System Algorithms

- Posted by Luba Belokon on July 28, 2017 at 4:00am
- Send Message View Blog

Today, many companies use big data to make super relevant recommendations and growth revenue. Among a variety of recommendation algorithms, data scientists need to choose the best one according a business's limitations and requirements.

To simplify this task, my team has prepared **an overview of the main existing recommendation system algorithms.**

Collaborative filtering

Collaborative filtering (CF) and its modifications is one of the most commonly used recommendation algorithms. Even data scientist beginners can use it to build their personal movie recommender system, for example, for a resume project.

When we want to recommend something to a user, the most logical thing to do is to find people with similar interests, analyze their behavior, and recommend our user the same items. Or we can look at the items similar to ones which the user bought earlier, and recommend products which are like them.

> *These are two basic approaches in CF: user-based collaborative filtering and item-based collaborative filtering, respectively.*

In both cases this recommendation engine has two steps:

1. Find out how many users/items in the database are similar to the given user/item.
2. Assess other users/items to predict what grade you would give the user of this product, given the total weight of the users/items that are more similar to this one.

What does "most similar" mean in this algorithm?

All we have is a vector of preferences for each user (row of the matrix R) and the vector of user ratings for each product (columns of the matrix R).

User / Item	Batman	Star Wars	Titanic
Bill	3	3	
Jane		2	4
Tom		5	

First of all, let's leave only the elements for which we know the values in both vectors.

For example, if we want to compare Bill and Jane, we can mention that Bill hasn't watched Titanic and Jane hasn't watched Batman until this moment, so we can measure their similarity only by Star Wars. How could anyone not watch Star Wars, right? :)

The most popular techniques to measure similarity are cosine similarity or correlations between vectors of users/items. The final step is to take the weighted arithmetic mean according to the degree of similarity to fill empty cells in the table.

Matrix decomposition for recommendations

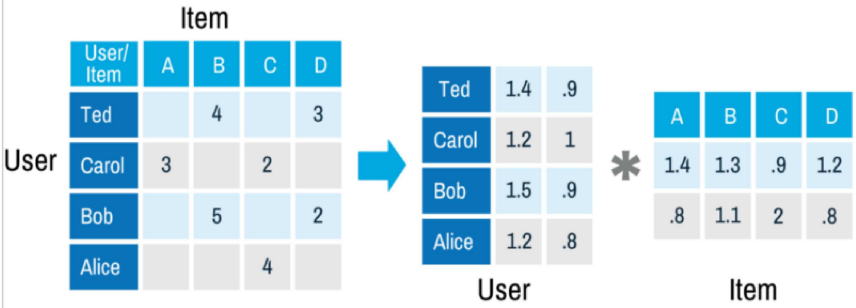
The next interesting approach uses matrix decompositions. It's a very elegant recommendation algorithm because usually, when it comes to matrix decomposition, we don't give much thought to what items are going to stay in the columns and rows of the resulting matrices. But using this recommender engine, we see clearly that **u** is a vector of interests of **i-th user,** and **v** is a vector of parameters for **j-th film**.

$$x_{ij} \approx \langle u_i, v_j \rangle$$

$$\sum_{i,j} \left(\langle u_i, v_j \rangle - x_{ij} \right)^2 \rightarrow min$$

So we can approximate **x** (grade from i-th user to j-th film) with dot product of **u** and **v**. We build these vectors by the known scores and use them to predict unknown grades.

For example, after matrix decomposition we have vector (1.4; .9) for Ted and vector (1.4; .8) for film A, now we can restore the grade for **film A−Ted** just by calculating the dot product of (1.4; .9) and (1.4; .8). As a result, we get **2.68 grade**.



Clustering

The previous recommendation algorithms are rather simple and are appropriate for small systems. Until this moment, we considered a recommendation problem as a supervised machine learning task. It's time to apply unsupervised methods to solve the problem.

Imagine, we're building a big recommendation system where collaborative filtering and matrix decompositions should work longer. The first idea would be **clustering**.

> *At the start of a business, there is a lack of previous users' grades, and clustering would be the best approach.*

But **separately, clustering is a bit weak**, because what we do in fact is we identify user groups and recommend each user in this group the same items. When we have enough data it's better to use clustering as the first step for shrinking the selection of relevant neighbors in collaborative filtering algorithms. It can also improve the performance of complex recommendation systems.

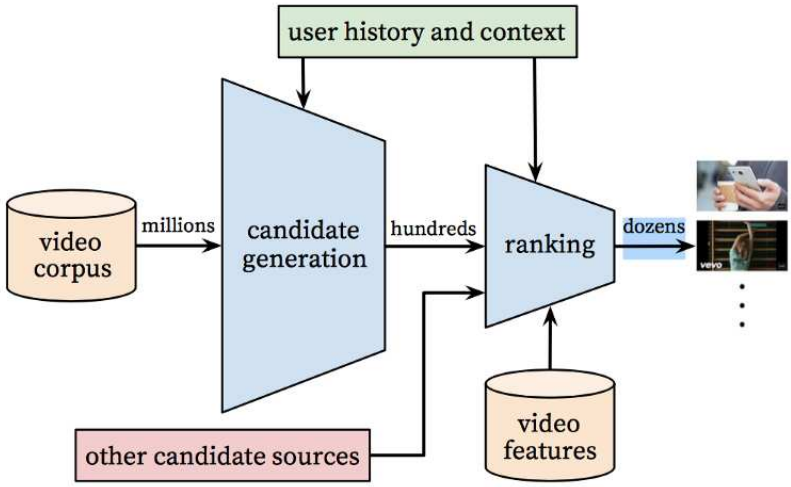
Each cluster would be assigned to typical preferences, based on preferences of customers who belong to the cluster. Customers within each cluster would receive recommendations computed at the cluster level.

Deep learning approach for recommendations

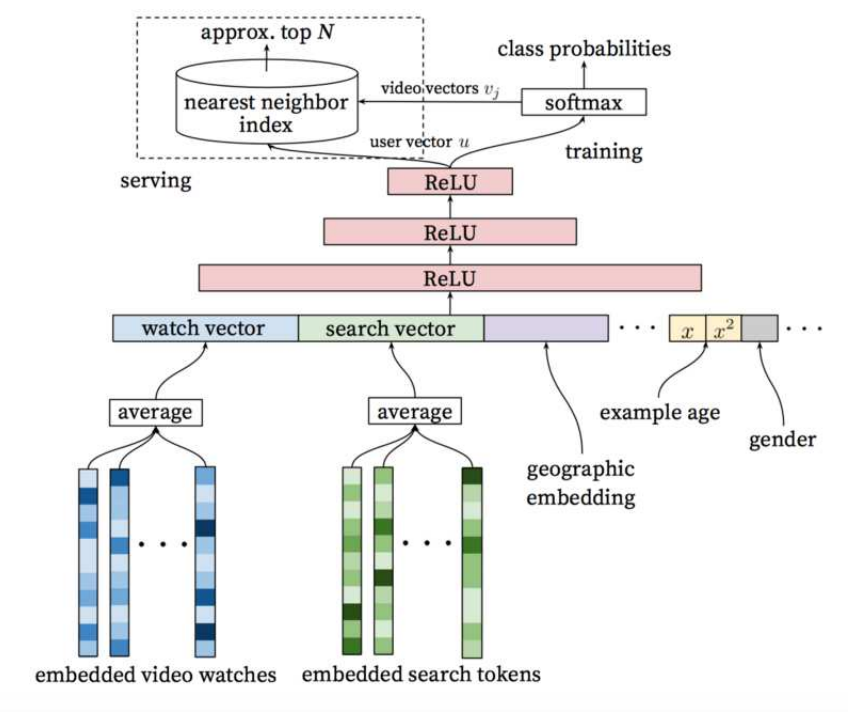
In the last 10 years, neural networks have made a huge leap in growth. Today they are applied in a wide range of applications and are gradually replacing traditional ML methods. I'd like to show you how the deep learning approach is used by YouTube.

Undoubtedly, it's a very challenging task to make recommendations for such a service because of the big scale, dynamic corpus, and a variety of unobservable external factors.

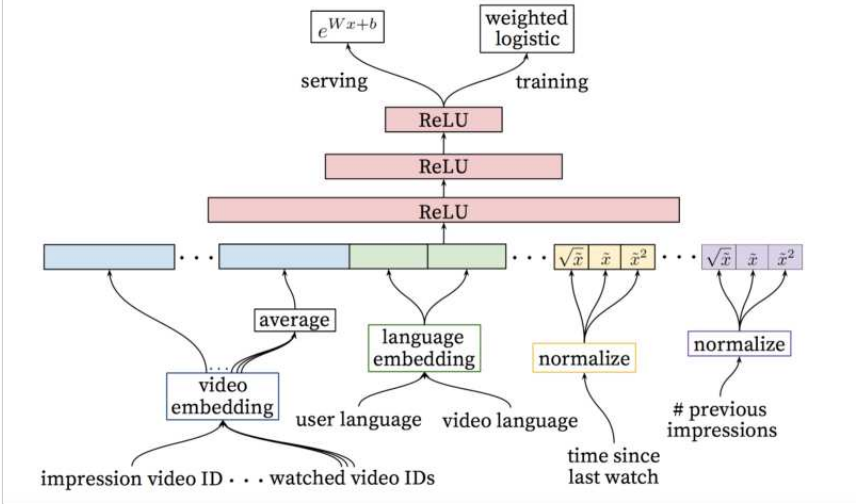
According to the study "Deep Neural Networks for YouTube Recommendations", the YouTube recommendation system algorithm consists of two neural networks: one for candidate generation and one for ranking. In case you don't have enough time, **I'll leave a quick summary of this research here**.



Taking events from a user's history as input, the candidate generation network significantly decreases the amount of videos and makes a group of the most relevant ones from a large corpus. The generated candidates are the most relevant to the user, whose grades we are predicting. The goal of this network is only to provide a broad personalization via collaborative filtering.



At this step, we have a smaller amount of candidates that are similar to the user. Our goal now is to analyze all of them more carefully so that we can make the best decision. This task is accomplished by the ranking network, which can assign a score to each video according to a desired objective function that uses data describing the video and information about users' behavior. Videos with the highest scores are presented to the user, ranked by their score.



Using a two-stage approach, we can make video recommendations from a very large corpus of videos while still being certain that the small number of them are personalized and engaging for the user. This design also enables us to blend candidates together that were generated by other sources.

$$P(w_t = i|U, C) = \frac{e^{v_i u}}{\sum_{j \in V} e^{v_j u}}$$

The recommendation task is posed as an extreme multiclass classification problem where the prediction problem becomes accurately classifying a specific video watch (wt) at a given time t among millions of video classes (i) from a corpus (V) based on user (U) and context (C).

Important points before building your own recommendation system:

- If you have a large database and you make recommendations from it online, the best way would be to divide this problem into 2 subproblems: 1) choosing top-N candidates and 2) ranking them.
- How do you measure the quality of your model? Along with the standard quality metrics, there are some metrics specially for recommendation problems: Recall@k and Precision@k, Average Recall@k, and Average Precision@k.
- If you are solving recommendation problems with classification algorithms, you should think about generating negative samples. If a user bought a recommended item, you should not add it as a positive sample, and others as negative samples.
- Think about the online-score and offline-score of your algorithm quality. A training model only on historical data can lead to primitive recommendations because the algorithm won't know about new trends and preferences.

This post originally appeared on here.