Recommender Systems 101 - a step by step practical example in R

Every one of us is unique! ...You are unique! there are *sooo* many people different from you... but at the same time, there are also A LOT that are damned similar to you... exhibiting the same behavior, interacting with the same people, liking the same things...

Whether you like it or not, it makes us extremely predictable and boringly *main stream*... But it's not necessarily a bad thing... You are already enjoying the benefits of the so called ***collective intelligence***, which is embedded in a lot of applications we use on a daily basis. Of course you are used to Facebook, Twitter or Linked.in suggesting people you might also know to expand your Social Media Network, or to Amazon pointing you to products that you might also consider when you are purchasing a particular item, or to Last.fm, Spotify & Co. suggesting you songs that are quite aligned with your musical taste... aren't you?

Well, all of them got something in common... the use of ***recommendation techniques*** to filter what *statistically* is most relevant for a particular user. In this post -a quite long one-, I'm going to cover the basics first to proceed with a step-by-step implementation of a recommendation engine.

A few basics first

Types of recommender systems

There are basically 2 approaches to make a recommendation... Let's say you want to recommend a set of additional products to a customer who purchased a product X:

- you can try to find out *what* in the product X was so attractive for the customer and suggest products having this "*what*"... We called them *Content based recommender systems.*
- you check for all other users who purchased product X as well, and make a list of other products purchased by these users... Out of this list, you take the products repeating the most. We called them *Collaborative filtering recommender systems*

For example, let's say I really liked "The Mission" and I gave the highest rating to this movie... The first type of systems might have modeled this movie as:

```
{actors: ["Robert De Niro", "Jeremy Irons"], director: "Roland Joffé", topics:["18th ce
ntury","Spanish colonization","Christian Evangelization"]}.
```

Based on that, movies from *Robert De Niro*, *Jeremy Irons* or *Roland Joffé* might be recommended, or movies like *"1492: Conquest of Paradise"* -Spanish colonization.
The second type of system -and the one imdb implements- will check in the database all users who rated *"The*

Mission" as high as I did and will retrieve all other movies rated high by these users... the list includes titles like *"Novecento"*, *"The innocent"* or *"The killing fields"*...

In this post we are going to implement a **Collaborative Filtering Recommender System**... In spite of a lot of known issues like the cold start problem, this kind of systems is broadly adopted, easier to model and known to deliver good results. Many implementations called **hybrid recommender systems** combine both approaches to overcome the known issues on both sides.

Tasks to be solved by RS

From the perspective of a particular user -let's call it *active user*-, a recommender system is intended to solve 2 particular tasks:

1. To *predict the rating* for an item or product, the user has not rated yet.
2. To create the *list of the top N recommended items*

In the step-by-step example you are going to see that you probably need both and the second one relies on the first one.

Validating Recommender Systems

Understanding how well a Recommender System performs the above mentioned tasks is key when it comes to using it in a productive environment.

The performance of the predictive task is typically measured by the deviation of the prediction from the true value. This is the basis for the Mean Average Error (MAE) or the squared version called Root Mean Square Error (RMAE)

In the formulas, *K* represents the set of all user-item pairings *(i, j)* for which we have a predicted rating *r^_ij* and a known rating *r_ij,* which was not used to learn the recommendation model. The basic idea behind these metrics is measuring the deviation between your predicted rated values and the real rated values over many users and items.

Implementing a Recommender System in R

Overview

One of the killer applications of Recommender Systems is the **conversion rate optimization**: customers find relevant products faster, cross-selling happens on a substantiated way and as a side effect, your image as a brand improves, as your attempt to be relevant for your customers is usually appreciated as value-adding, which also positively impacts the customer loyalty. That's why we are going to focus on this use case :)

We are not going to implement everything from scratch (thank you Captain Obvious!)... There are a few R packages implementing collaborative filtering engines, but I like recommenderlab the most.

1- Data Gathering

Sometimes the discovery of the affinity of users for certain items is not as straight forward as a data base with ratings. Yet, there are countless indications we can use to model this affinity.

Let's think of the clickstream data. You can measure referring method, page visited, clicked items, items part of a comparison, items part of the basket and checkout process, etc... You can even combine it with on-page indicators, like time-on-page (see the riveted time spent plugin for Google Analytics) or mouse moves (e.g.: with ClickTale, Mouseflow , etc). An orthogonal dimension is the timely aspect, for example how long ago was which interaction.

As a result, you have a lot of sessions with a lot of events for an user with respect to an item. A handy approach would be computing the User-Item Affinity per session taking also into account the Frequency and Recency, even with an overly simplified approach like the one below:

An then aggregating it for all sessions where this particular user had interactions with this particular item

2- Data Normalization

Sometimes the discovery of the affinity of users for certain items is not as straight forward as a data base with ratings. Yet, there are countless indications we can use to model this affinity.

Our code in R looks like:

```
library("recommenderlab") # Loading to pre-computed affinity data affinity.data
```