Recommendation Systems: Principles and practice

BY CHITTARANJAN TRIPATHY AND YANNIS PAVLIDIS

Recommendation systems are *sine qua non* for ecommerce, social networks, and media streaming service providers [1]. Product search—browsing through the product taxonomy graph or entering a search query to retrieve relevant products—is often the first hook for the customer to locate a product in e-retailer's inventory. The recommendation system then takes over and acts as a friend to guide the customer throughout the shopping journey by recommending the most relevant products the customer may buy in every step, without the customer having to search for them explicitly.

Given that we are submerged in an ever-growing amount of information, having a recommendation system that sifts through information and presents the most relevant pieces helps overcome information overload, and short circuits the shopping journey by helping customers find what they are looking for faster.

The past two decades have seen a significant amount of research on recommendation systems. Large state-of-the-art recommendation systems have been built in various domains, for example, Walmart's grocery offerings, Netflix's movie choices, Amazon's product suggestions, and YouTube's video recommendation systems. It is no exaggeration to say that recommendation systems are pervasive in our daily lives through the gadgets we use. The news we read on our smartphones, or the jobs suggested by LinkedIn, all come from recommendation systems.

In this article, we discuss the underlying principles and technologies behind a recommendation system. We also describe the key principles behind building a recommendation system with example code snippets. We will use an e-retailer, such as Walmart, as an example in our discussion. There are three parties to a recommendation system: customers (or users), products, and the recommendation system itself. Based on how a recommendation system views and interprets the customer-product interactions, two broad categories of recommendation systems have emerged.

The first one is the content-based recommendation system, which recommends similar items the user has liked in the past. The similarity of the items is determined by their properties, also known as features. Here are the steps involved:

(1) For each item, represent it as an item feature vector, called the "item profile."

(2) Find similar items based on feature vector matching.

(3) Represent each user as a feature vector, called the "user/customer profile" that represents the user's preferences.

(4) Recommend items that correlate the most with the user profile through another similarity measure.

While the main advantage of a content-based approach is that it does not need other user data to do a similar item search, thereby avoiding the so-called new item or item cold start problem, on the flip side lies its inability to recommend to new users whose profile is not known.

The second class of recommendation systems is based on collaborative filtering. The collaborative filtering algorithm recommends to a user those items that are preferred by similar users. Therefore the process is a collaborative one, in which a group of similar users help each other in making good recommendations in an anonymous but collaborative manner. While collaborative filtering is widely used and works on any item set, IT nevertheless suffers from cold start (new user), data sparsity (users only express opinion such as like or bought on a limited subset of items), non-recommendation of unrated or unbought items, and bias toward best-selling or popular items.

Here we discuss a hybrid approach that uses content-based methods with collaborative filtering. It is called "item-to-item collaborative filtering" [2, 3].

The idea here is for every item *i* we find the item *j* that was bought more frequently by the users who also bought *i*. Once the item-to-item similarity matrix is built—an expensive computation often done offline—the recommendation list generation is a real-time search of the similarity matrix to compose a recommendation list.

ITEM AND USER REPRESENTATION

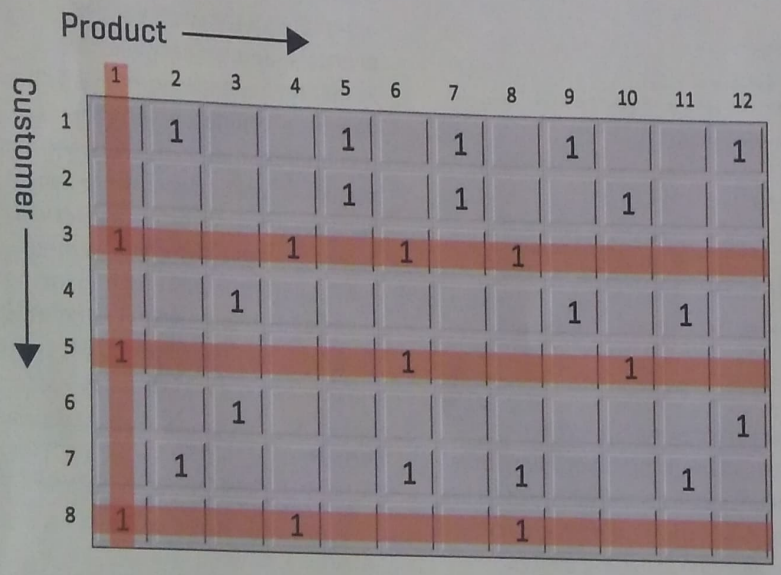
How do we represent items and users? We describe them as vectors of their properties, called attributes or features. We use the terms "item profile" and "user/customer profile," respectively, to refer to an item and a user.

An item can be a product sold online, a news article, an audio song, a video, an image, a restaurant, or other location of interest. The item profile for an item represents a collection of important features of that item, which can be used for identification. For example, a fruit is represented by the profile containing the following features: category, brand, is-organic, shelf-life, price, and is-tropical. More features, such as seasonal availability, color and texture of the fruit, production location, nutrition facts, and allergy information, can be used depending on the context of usage. As we notice, the features can be real numbers, Boolean, or multi-class features.

Listing 1. Similarity measures for item/customer vectors.

```
import numpy as np
from numpy.linalg import norm
from math import sqrt

def dist_manhattan(v1, v2):
    assert len(v1) == len(v2), "The dimensions of vectors do not match."
    return sum(abs(v1[i] - v2[i]) for i in range(len(v1)))

def dist_euclidean(v1, v2):
    assert len(v1) == len(v2), "The dimensions of vectors do not match."
    return sqrt(sum((v1[i] - v2[i])**2 for i in range(len(v1))))

def sim_cos(v1, v2):
    assert len(v1) == len(v2), "The dimensions of vectors do not match."
    return np.inner(v1, v2) / (norm(v1) * norm(v2))

def sim_jac(v1, v2): # Here v1, v2 are lists of different dimensions
    v1s, v2s = set(v1), set(v2)
    return len(v1s.intersection(v2s)) / len(v1s.union(v2s))
```

Figure 1. Item 1 was bought by customers 3, 5, and 8, who also bought items 4, 6, 8, and 10. Therefore, similarities between item 1 and each item in the set {4, 6, 8, 10} will be computed (in total four) as opposed to 11 pair-wise similarity computations for item 1.



Similarly, we characterize users by their user profiles. Part of the profile can be declared by the user, such as gender, age segment, or living locations, and (optionally) declared statements such as food allergies and clothing preferences.

Other features often generated are purchase-based, such as item types, quantities bought in the past, and the amount spent, and browse-based such as time spent on product detail pages, clicks, device type for browsing, etc.

The user profile captures user interactions with the items, plus the overall ecommerce website that includes both purchase and non-purchase sessions. Finding good features is an important task that often requires expertise in the domain, but when done properly it leads to improved recommendation quality and performance.

Now that we know how to represent an item and a user, the next thing is to explore how we can quantify item-item similarity and user-item affinity using similarity measures.

SIMILARITY MEASURES

Similarity between two objects defines how closely they are related. For example, let's assume two items i_1 and i_2 are represented by item profile vectors in m=3 dimensions with numeric features $\overrightarrow{i_1}$ =[2,3,5] and $\overrightarrow{i_2}$ =[3,4,4], respectively. A distance function, such as Manhattan distance or Euclidean distance, is used to compute the distance between i_1 and i_2. These distance functions are defined below in Equations 1 and 2, and the code is provided in Listing 1.

$$(Manhattan)\quad d_1(\vec{i_1},\vec{i_2}) = \sum_{k=1}^{m} |x_k(i_1) - x_k(i_2)| = 3. \tag{1}$$

$$(Euclidean)\quad d_2(\vec{i_1},\vec{i_2}) = \sqrt{\sum_{k=1}^{m} (x_k(i_1) - x_k(i_2))^2} = 1.73. \tag{2}$$

Another measure of similarity is the angle between $\overrightarrow{i_1}$ and $\overrightarrow{i_2}$, the smaller the angle is, the more similar the two items are. This is called the "cosine similarity" derived from the simple dot-product formula of vectors and is defined by Equation 3. The code is provided in Listing 1.

$$(Cosine)\ sim\_cos(\vec{i_1},\vec{i_2}) = cos\angle\vec{i_1},\vec{i_2} = \frac{\vec{i_1}\cdot\vec{i_2}}{||\vec{i_1}||\cdot||\vec{i_2}||} = \frac{38}{6.16\times 6.40} = 0.96. \tag{3}$$

When the i_1 and i_1 represent documents using bag-of-words or set $S[\cdot]$ representation, a different similarity measure called Jaccard similarity is used as defined in Equation 4, and the code is in Listing 1. It is nothing but the size of the intersection to the size of the union of i_1 and i_2.

$$(Jaccard)\ sim\_jac(\vec{i_1},\vec{i_2}) = \frac{S(\vec{i_1})\cap S(\vec{i_2})}{S(\vec{i_1})\cup S(\vec{i_2})} = \frac{|\{3\}|}{|\{2,3,4,5\}|} = 0.25. \tag{4}$$

Note that Jaccard similarity does not require $\overrightarrow{i_1}$ and $\overrightarrow{i_2}$ to be of same dimension.

ITEM-TO-ITEM COLLABORATIVE FILTERING ALGORITHM

For each user, an algorithm matches purchased and rated items to similar items. This combined set of similar items is used to generate a ranked list of recommendations. The algorithm works in two stages.

The first stage is the item similarity matrix computation, which is a giant matrix with entry for each item pair. Computing such a giant matrix is an expensive task—both time and memory usage-wise. A key observation that helps here is most item pairs have no common customers, hence the matrix is a

sparse one [see Figure 1]. Therefore, computing the similarity only for those item pairs with common customers tremendously reduces the time and space complexity.

If each item is represented as a vector in d dimensions, and there are n items in the catalog, then the giant matrix computation would take $O[n^2d]$ time. However, with the observation on sparsity of the item, time complexity is closer to $O[nd]$ in practice.

The similarity matrix computation is done offline. The item-item similarity measure can be any suitable metric; often cosine similarity is used. Listing 2 shows an example implementation of the item similarity matrix computation.

The second stage of the algorithm is the generation of a ranked list of recommendations. For a user, this computation is done online by doing lookups of the item similarity matrix, retrieving the similar items, and ranking them using item similarity scores, popularity of items, customer's recent purchases, current shopping and browsing context of the customer such as items in the shopping cart, and the current product detail page the customer views. This step can be implemented as a simple weighted sum of ratings procedure or as a sophisticated machine learning based ranking model, keeping in mind that this step must be a low-latency one. In Listing 2 we implement a very simple-minded ranked list generator that only considers the item similarity scores.

The item-to-item collaborative filtering algorithm has several key advantages over the traditional content based and collaborative filtering algorithms, including high-scalability and excellent recommendation quality. Further

optimizations are often achieved by coupling this algorithm with explore-exploit based mechanisms.

CONCLUSION

In this article, we presented the key components of one of the most popular and highly scalable approaches to recommendation system design: item-to-item collaborative filtering. In recent times, a large number of new techniques have been developed that heavily use machine learning and deep learning methods. While the current research makes an effort to make the recommendations explainable, fair, and privacy-preserving, another line of research focuses on quality of recommendations including addressing cold-start, novelty, serendipity, unexpectedness, diversity, and coverage issues. For instance, during grocery basket recommendations the recommendation is not focused on a single item but on the whole basket. Also, explore-exploit frameworks to balance between immediate intent and long-term customer engagement have been an integral part of modern recommendation systems, which we will present in a future issue.

References

[1] Agarwal, D. K., and Chen, B.-C.. *Statistical Methods for Recommender Systems*, 1st Edition Cambridge University Press, Cambridge, 2016.

[2] Sarwar, B. , Karypis, G., Konstan, J., and Riedl, J. Item-based collaborative filtering recommendation algorithms. WWW'01, 2001, 285–295.

[3] Linden, G., Smith, , B., and York, J. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing* 7,1 (2003), 76–80.