16ML Notes: Collaborative Filtering

Abinav Routhu, Maxwell Chen

December 11, 2020

1 Introduction

So far, you've heard a lot about regression and classification in the context of making predictions and classifications. These are two storied problems in statistics that have become the bedrock of machine learning education. In this note, we take a sidestep from the rigorous statistical underpinnings of machine learning, and instead look at a highly-applied use case for machine learning in today's world.

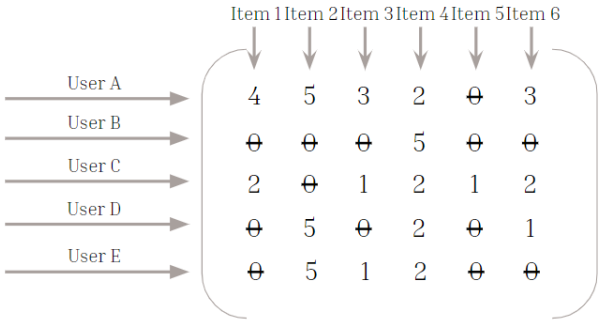
Recommendation Systems, also known as Recommender Systems, are models that aim to predict and suggest products or items that a specific user will likely use or buy. For example, this might be recommending a similar song on a music streaming service like Spotify, recommending products on Amazon similar to your latest purchases, or recommending enticing ads to users on YouTube. Mastering the art of learning from user data to predict user interest in products to a high degree of accuracy is very useful to companies: for instance, 80% of movies watched on Netflix come from recommendations, while 60% of video clicks came from home page recommendations on YouTube [1]!

There are 2 fundamental paradigms in building Recommendation Systems: **content-based methods** and **collaborative filtering**. Content-based systems rely on knowledge about the items aka products and the users in order to make recommendations. For instance, Netflix might recommend cartoons to users who are under 10. Collaborative filtering, on the other hand, doesn't require this *a priori* information (information needed before/prior to analysis). This approach simply compares item preferences between users; the idea here is that users who share opinions about a few items will generally share opinions about all items. For instance, if users Sean and Michelle both loved *Titanic*, and Michelle also loves *Mamma Mia*, then Netflix may recommend Sean *Mamma Mia* as well.

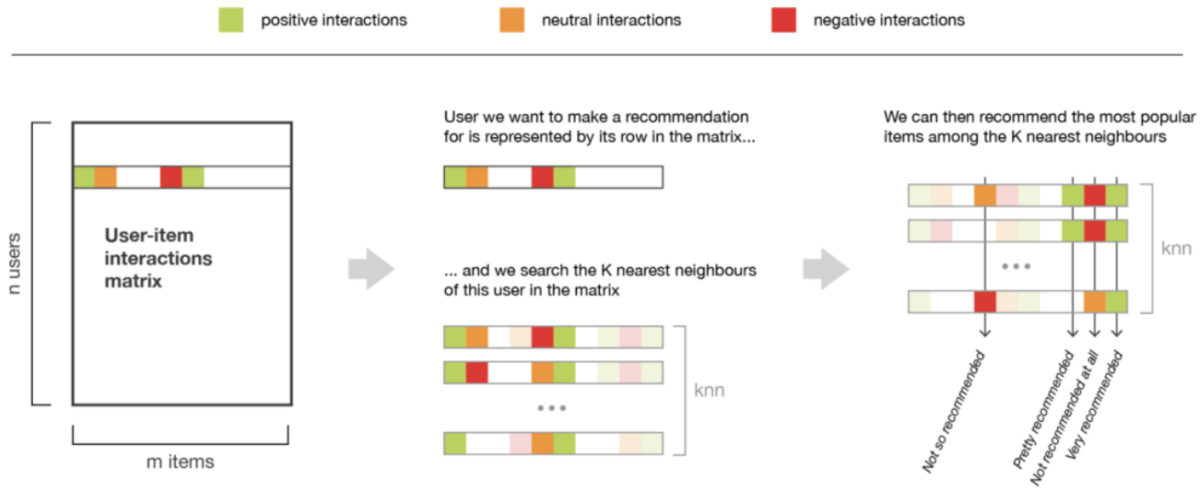
Furthermore, systems utilizing both paradigms are called hybrid systems. The content-based approach should feel familiar – using appropriate featurizations of data for prediction is what we've been doing this whole time. In order to hammer home the fundamentals and key ideas, we will limit our scope to collaborative filtering in this note.

2 Memory

Our central tool of analysis will be the **user-item interaction matrix**. For m users and n items, this matrix $A \in \mathbb{R}^{m \times n}$ has entry a_{ij} representing the i^{th} user's interaction with the j^{th} item. This interaction may be a rating, a ranking, etc.



One natural step might be to separate users into groups utilizing group preferences to guide recommendations – clustering! If we focus on just the i^{th} user, we can search for the k users most similar to the i^{th} user based on the items they have both interacted with. Then, we find the items rated best by this subgroup of users. From this list, the i^{th} user is recommended the top items they have not yet interacted with.

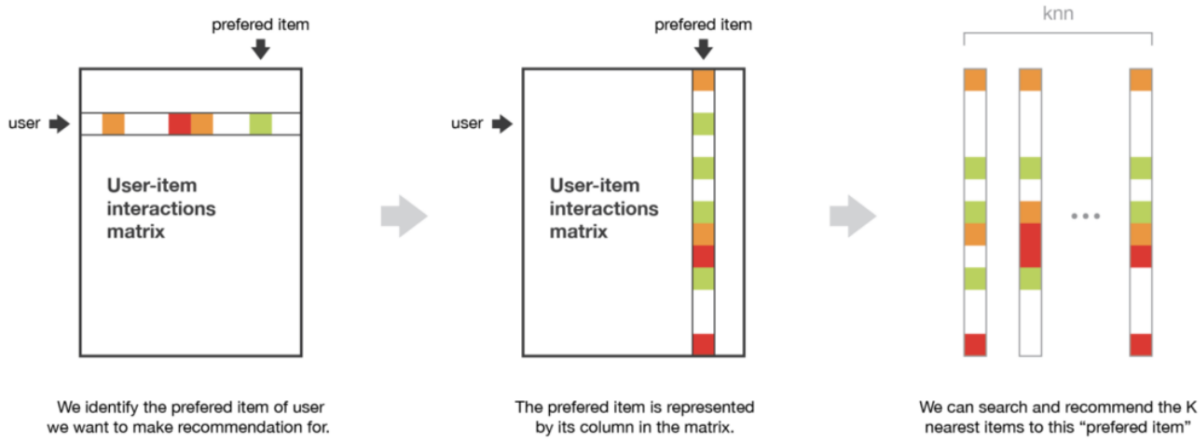


Towards Data Science: Introduction to recommender systems

This process is essentially $k-$nearest neighbors; it's conceptually simple and reliable. However, this process, the **user-user approach**, is highly dependent on having a wide breadth of interactions recorded for each user. In general, the interaction matrix is highly

sparse. Changing just a few entries in A can radically impact the outputs produced. Another approach is the **item-item approach**, where clusters are based on items instead of users.

For the item-item approach, start by instead picking the i^{th} user's favorite item, the j^{th} item. Then, search for the k items most similar to that j^{th} item. Repeat for the i^{th} user's top items. Recommend the items that were most frequently neighbors of these favorite items.



Towards Data Science: Introduction to recommender systems

These procedures are considered **memory-based** since they exclusively work with the data and cast no assumptions about how it is generated or related. But how do they stack up against each other? While users tend to only ever interact with a few items, a given item generally has interacted with a decent subset of users. This, comparatively, *reduces the variance* of the item-item approach. But, this process is also *more general*. Using only m favorite items uses less information specific to the i^{th} user leading to less customized recommendations overall. There are more issues in addition to the sparsity problem. For instance, if the similarity measure is just Euclidean distance, then a user that agrees on a single mutual item interaction is closer than a user that agrees on 29 out of 30 mutual item interactions. Furthermore, these approaches are biased toward items that have more interactions. These popular items are more frequently recommended and thus interacted with, becoming even more popular while unpopular items fade to obscurity[2]. At an extreme, for new users and items with *no* recorded interactions, the system has no way of incorporating them into the recommendation scheme. In the literature, this is known as the **cold start problem**.

Lastly, the *knn* algorithm itself scales poorly. Searching for k neighbors out of n users with m items takes $O(nmk)$ time. Getting more efficient and regularizing these biases in our methodology will require developing models with innate structure.

3 Model

Fundamentally, the goal at hand is to fill in the missing values of the user-item interaction matrix A given the present entries. Even though A tends to be very large, these entries are not random. Most of the data should be explainable by few, simple patterns. We formalize this intuition by assuming the ratings are *generated by a low dimensional latent space.*

A can be seen as the product of 2 matrices, $U \in \mathbb{R}^{m \times l}$ and $V \in \mathbb{R}^{l \times n}$, where l is the dimension of the latent space. This is **matrix factorization**. Conceptually, each item corresponds to a column of V and is described by l features. U, conversely, has a row for each user where each of the l values states the user's affinity for that feature.

Imagine A contained movie ratings and it was generated by a 2 dimensional latent space. Then, each movie (item) can be described by exactly 2 features, say, its length and how serious it is. Every row of U would then be 2 values, the first measuring that user's length preference, the second measuring the user's preference for humor in movies. The columns of V are **item embeddings** and the rows of U are **user embeddings**.

Okay, so how to find U and V? Imagine we *did* know every entry in A. Then, the *Eckart-Young* Theorem says the best rank-l approximation is simply given by the SVD of A. Can you see why? This fact inspires the algorithm that follows which is why it is sometimes called the "SVD" method, even though the actual SVD is never used. Confusing, right?

We don't know A fully, but at least we know some entries. Let's call the set of entries a_{ij} that we do know K, the known set. Then, using gradient descent, we can find U and V that minimize the distance between a_{ij} and $\hat{a}_{ij}$, our estimation for that entry.

$$\hat{a}_{ij} = \vec{u}_i^T \vec{v}_j$$

$$\min_{U,V} \frac{1}{2} \sum_{a_{ij} \in K} (a_{ij} - \hat{a}_{ij})^2$$

However, this model is very prone to overfitting and has many free parameters (defining U and V means there are $l(m+n)$ free parameters). To resolve this, we can penalize our parameters. Another improvement we can make is to not estimate a_{ij} directly, but instead its deviation from average.

Imagine you are estimating the i^{th} user's rating for the j^{th} item. A reasonable place to start would be to simply guess the average rating across all ratings seen so far, call this μ. But, suppose you also know that the i^{th} user tends to rate all items very generously; for any item, he rates far above the average. Conversely, you know the j^{th} item is a poor product; it tends to elicit negative ratings far below the average. We should adjust our guess by this information, b_i^u and b_j^v, the user offset and item offset respectively. Only then, finally, do we consider the i^{th} user's specific affinity for the j^{th} item. This intuition is formalized in the model below.

$$\hat{a}_{ij} = (\mu + b_j^v + b_i^u + \vec{u}_i^T \vec{v}_j)$$

$$\min_{U,V,\vec{b^u},\vec{b^v}} \frac{1}{2} \sum_{a_{ij} \in K} (a_{ij} - \hat{a}_{ij})^2 + \lambda((b_j^v)^2 + (b_i^u)^2 + \|\vec{u_i}\|_2^2 + \|\vec{v_j}\|_2^2)$$

To recap, K is the set of all known ratings $\{a_{ij}\}$. μ is the global average rating computing by simply averaging over K. $\vec{u_i^T}$ and $\vec{v_j}$ are the embeddings for the i^{th} user and j^{th} item, respectively. b_j^v and b_i^u are their respective biases representing how much the average rating of the j^{th} item exceeds μ and how much the average rating given by the i^{th} user exceeds μ. λ is the ridge coefficient, a hyperparameter describing the severity of the weight penalty.

$$\begin{cases} e_{ij} = a_{ij} - \hat{a}_{ij} \\ b_j^v \leftarrow b_j^v + \gamma(e_{ij} - \lambda b_j^v) \\ b_i^u \leftarrow b_i^u + \gamma(e_{ij} - \lambda b_i^u) \\ \vec{u_i} \leftarrow \vec{u_i} + \gamma(e_{ij}\vec{v_j} - \lambda \vec{u_i}) \\ \vec{v_j} \leftarrow \vec{v_j} + \gamma(e_{ij}\vec{u_i} - \lambda \vec{v_j}) \end{cases}$$

The updates above with learning rate γ are easily derived given the convexity of the objective (a concept you may explore later in classes like EECS 127). Now, we can use techniques like gradient descent and stochastic gradient descent to find optimal parameters for the matrix factorization. This is Funk's celebrated "SVD" Matrix Factorization Algorithm he made to win 3rd in the 2006 Netflix Prize, a challenge to find the best collaborative filtering algorithm with a $1M prize[3]!

4 State of the Art

The aforementioned Netflix Prize was a competition run from 2006 to 2009 to improve their recommendation algorithm. The prize attracted many top researchers and led to exciting developments in recommendation systems e.g. ensemble methods, "SVD", "SVD++"[4]. In the decade since, the field has turned to methods in Deep Learning to push the forefront of research (the premier conference in the industry, RecSys, has offered Deep Learning workshops since 2015). These methods offer end-to-end approaches in easily composable models that are better at learning complex, nonlinear representations of multi-modal interaction data i.e. generating superior embeddings with less restrictions on the input data and the patterns observed. (Note the "SVD" algorithm presented above is strictly linear.)

The authors highly recommend the survey of *Zhang, Yao, Sun, and Tay 2018*[1] to the interested reader.

References

[1] Aixin Sun Shuai Zhang, Lina Yao and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys*, 1:1–35, 2018.

[2] Baptiste Rocca. Introduction to recommender systems. https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada, 2019.

[3] Simon Funk. Netflix update: Try this at home. https://sifter.org/simon/journal/20061211.html, 2006.

[4] Yehuda Koren. The bellkor solution to the netflix grand prize. Published on Netflix Prize Forums, 2009.