

16ML Notes: Collaborative Filtering

Abinav Routhu, Maxwell Chen

December 2, 2020

1 Introduction

By now, you’ve learned much about regression and classification, 2 classic problems in statistics that gave rise to the study of machine learning. In this section, we turn away from the statistical background of machine learning and discuss an important context for machine learning in today’s world.

Recommendation systems are models that seek to suggest the (heretofore unseen) products best befitting a specific user. For example, this might be recommending a similar song on a music streaming service or matching viewers with the most pertinent ads. Being able to customize user experience and predict user interest to a high degree can be incredibly useful; 80% of movies watched on Netflix came from recommendations, 60% of video clicks came from home page recommendation in YouTube [1]!

There are 2 fundamental paradigms in recommendation systems: **content-based** and **collaborative filtering**. Content-based systems rely on knowledge about the items (products) and the user to make recommendations e.g. Netflix might recommend cartoons to users under 10. Collaborative filtering, on the other hand, eschews the need for this *a priori* information by simply comparing item preferences between users; the idea here is that users who’ve shared opinions about a few items will generally share opinions about all items e.g. given both Sean and Michelle loved “Titanic”, Netflix may recommend Sean “Mamma Mia” since Michelle also adores “Mamma Mia”.

Additionally, systems utilizing both paradigms are called hybrid systems. The content-based approach should feel familiar, using apropos featurizations of data for prediction is what we’ve been doing this whole time. We’ll focus instead on collaborative filtering.

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. For simplicity, suppose +1 for *positive* interaction, -1 for *negative*, and 0 for no recorded interaction.

One natural step might be to separate users into groups utilizing group preferences to guide recommendations – clustering! Focusing on just the i^{th} user, search for the k users most similar to the i^{th} user based on the items they have both interacted with.

Then, find the items viewed most positively by this subgroup of users. Recommend the top n items that the i^{th} user has not yet interacted with.

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 available. 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. These procedures are considered **memory-based** since they exclusively work with the data and cast no assumptions about how it is generated or related.

Instead, start by picking the i^{th} user most positively reviewed item, the j^{th} item. Then, search for the k items most similar to the j^{th} item. Repeat for the i^{th} user's top m items. Recommend the n items that were most frequently neighbors of the m favorite items.

How do these approaches stack up? 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].

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 is why this algorithm is sometimes called the SVD method, even though the SVD is never used. But what we can do is to try to optimize what is known.

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

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

However, this model is very prone to overfitting – defining U and T means there are $l(m+n)$ free parameters. Instead of estimating a_{ij} , it helps to estimate deviation from average and to penalize the parameters to prevent them from growing without bound.

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

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

To recap, K is the set of all known ratings $\{a_{ij}\}$. Compute μ 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 .

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

The updates above are easily derived given the convexity of the objective. Now, we can use SGD 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.