Alternating Least Squares in the Context of Recommender Systems and their Applications

Introduction

The aim of many online businesses today is to improve customer experience by personalizing their experience. This might involve specific, tailored recommendations on e-commerce websites like Amazon and Costco, or movie and TV show recommendations on streaming websites like Netflix, Hulu, and HBOMax. Further, online music streaming applications like Apple Music and Spotify may create tailored playlists for users' based on past listening history on the platform. Regardless of the application, personalized recommendations play a large role in *something about retaining customers and keeping them engaged or making them buy more stuff* and have become common practice amongst online applications.

The vast majority of recommendations are made in real time, and can be based on explicit user ratings of items, past purchase history, social media comments and searches, clicks, and streams of video content, amongst many others. However, depending on what kind of user feedback is gathered, implicit or explicit, the user-item preference must be modeled differently.[1] Given explicit feedback, such as a movie rating a user has given a film, one can model this item-user relationship explicitly on the basis of the ratings, and make recommendations directly from this data.

More difficult is the case of implicit data, which requires a system to infer a user's preference towards an item from indirect information,[2] such as the amount of time they spend looking at particular items on amazon, mouse clicks, number of listens of a song, and the presence or absence of events.[3] Thus, it is up to the recommender system to not only predict future recommendations, but also to infer how prior user behaviour indicates preference of a user to a particular item.

In this paper, we will investigate past recommender system algorithms, and how implicit and explicit data should be handled differently. We will focus on the Alternating Least Squares (ALS) algorithm for recommendations given implicit user data and its underlying mathematical derivations. We demonstrate the efficiency of the ALS algorithm in making these predictions, and possible future improvements and areas of study.

---

[1] https://dl.acm.org/doi/abs/10.1145/2365952.2365972
[2] ibid
[3] https://dl.acm.org/doi/pdf/10.1145/2365952.2365972

Preliminaries and Notation

In this paper, we will refer to items as the content whose attributes are used in the recommender models,[4] which are being recommended to users. This could be items in an amazon catalog, songs in Spotify, and movies on Netflix, amongst many others.

We will refer to attributes as characteristics of items. We will refer to attributes or characteristics of an item as the factors or features of an item.

Explicit user data is data from which we can explicitly infer both a user's like and dislike towards an item. For example, a rating from 1 to 5 can directly inform a system about how much a user likes or dislikes an item - a rating of 5 would indicate a high preference toward the item, while a 1 would indicate dislike. The absence of a rating simply indicates that a user's inclination towards that item is unknown. This can include ratings, a thumbs up or down, or a review. This data is very informative but often unavailable or only available in small quantities, which is why implicit data is often used in recommender systems.

Implicit data is data gathered about a user's interaction with a system, application, or catalog. This can be how many times a user listened to a song or watched a movie, their purchase history on an e-commerce website, or click patterns. Implicit data is abundant; it is data gathered about how users interact with an application, and thus requires no additional input from them and is readily available. The problem this poses is its interpretation. While a user watching a movie many times gives some indication of preference, this type of data does not tell us anything about a user's dislike. If they only watch a movie once, this could mean that they did not like it very much, but it also could be the case that this is their favourite movie and they have simply seen it already. Further, a lack of data about an item could indicate that a user dislikes it, or that they simply have not interacted with it before.

Content-Based Approaches in Recommender Systems

Content-based recommender systems, or more simply, content-based filtering, was an early successful approach used in predicting recommendations for users. Content-based recommender systems can work on both implicit and explicit data, and although these cases are handled differently, they all work by creating recommendations for a user based on that same user's past interactions with items. A system will gather data about how a user interacts with items in an application (i.e. movies on Netflix) and try to predict, based on this prior information and the similarity of unseen items to their prior preferences, which unseen items a user will react positively to (a recommendation).[5] Given more new data about the user, the system can improve its recommendations.

---

[4] https://www.analyticsvidhya.com/blog/2015/08/beginners-guide-learn-content-based-recommender-systems/

[5] https://towardsdatascience.com/introduction-to-recommender-systems-1-971bd274f421

Importantly, a content-based system requires a profile of sorts about a user to be built up before it can begin generating recommendations.[6] For example, given an e-commerce website like Amazon, the system might create a user profile that includes attributes of previously purchased or viewed items. If the user has purchased many pottery tools and supplies, their profile might include keywords like “clay”, “ceramics”, “crafts”, etc. After the profile is created, the system might recommend other items associated with those same keywords. A profile might also be created by asking the user to select categories they are interested in when they create an account. There are many ways to hand-engineer features which can be based on different types of data (i.e. implicit or explicit).

A common way to achieve this can be illustrated with an example. Suppose we have a few users in an application store (i.e. Google Play, Apple App Store, etc.) and the applications are associated with categorical words.[7] This is similar to one-hot encoding, where a 1 indicates that the application contains that attribute, and 0 implies that it does not. Similarly, we encode the users with the same categorical words, but this time a 1 implies that the user has an affinity to this category, and a 0 means they do not. This can be illustrated in the tables below:

Categories for Apps

	Health	**Game**	**Lifestyle**	**Science**	**…**	**Education**
App 1	1	0	1	1		0
App 2	1	1	1	1		0
App 3	0	1	1	0		0
App 4	1	0	0	1		1

Category Preferences of Users

	Health	**Game**	**Lifestyle**	**Science**	**…**	**Education**
User 1	1	0	1	0		0
User 2	0	1	0	1		1

Given this information, the aim of a content-based recommender system is to find similar items based on items that a particular user has liked in the past. It is important to note that in this type of filtering, other users’ do not influence recommendations for any other user, and vice versa. The items in the catalog are transformed into vectors of their features in an n-dimensional space, where n is the number of features/categories (the feature space). The same is done for the users, where each user is represented as a vector in the same n-dimensional space.

---

6 https://www.upwork.com/resources/what-is-content-based-filtering#:~:text=Content%2Dbased%20filtering%20is%20a,them%20to%20a%20user%20profile.

7 https://developers.google.com/machine-learning/recommendation/content-based/basics

Should I talk about user/item content matrices here?

To perform the actual recommendation portion of content-based filtering, a measure of similarity is used. Cosine similarity or vector dot products are commonly used metrics, but note that cosine similarity will not account for magnitude if used, and data should be normalized as necessary. Given that all users and items are represented as vectors in the same n-dimensional space, a measure of similarity between a user and items can then be taken, and the most similar items will be recommended. It is also possible to find other similar items to a given item that a user may have purchased. A dot product similarity measure for a user vector to an item vector is show below:

** Insert equation here.**

While this approach was successful in generating recommendations based on a user's other preferences, it does not leverage the information that we gain from other users with similar preferences. In other words, content-based filtering does not use information about other users to generate recommendations for a user. Further, this method requires a priori information about a user's preferences (a user profile), as well as categorized items with the same key words. While optimizations are possible, this approach requires a lot of data manipulation and preparation which is both cost and time intensive. Collaborative filtering successfully uses information about both a single user's past interactions with items, and other similar users interactions with items, all without requiring hand-engineered features to predict a recommendation.

Previous Collaborative Filtering (CF) Approaches in Recommender Systems

In response to the limitations of content-based filtering, collaborative filtering (CF) was developed. Collaborative filtering is an umbrella term for a group of models and algorithms that are used to predict unknown user-item ratings, but what they all have in common under the umbrella of CF is that they only use already observed ratings (implicit or explicit) to generate a recommendation.[8] It uses similarities between users and items simultaneously to generate recommendations, without the need of manually constructed features.[9]

Collaborative filtering also provides increased capabilities to a recommender system; instead of only recommending similar items to a user based on previous preference data, CF also uses unlooked-for relationships between users to predict recommendations. For example, an item might be recommended to user X if a similar user Y liked that item. Differently, an item X might be recommended to a user if it is similar to item Y, an item that the user already "liked".

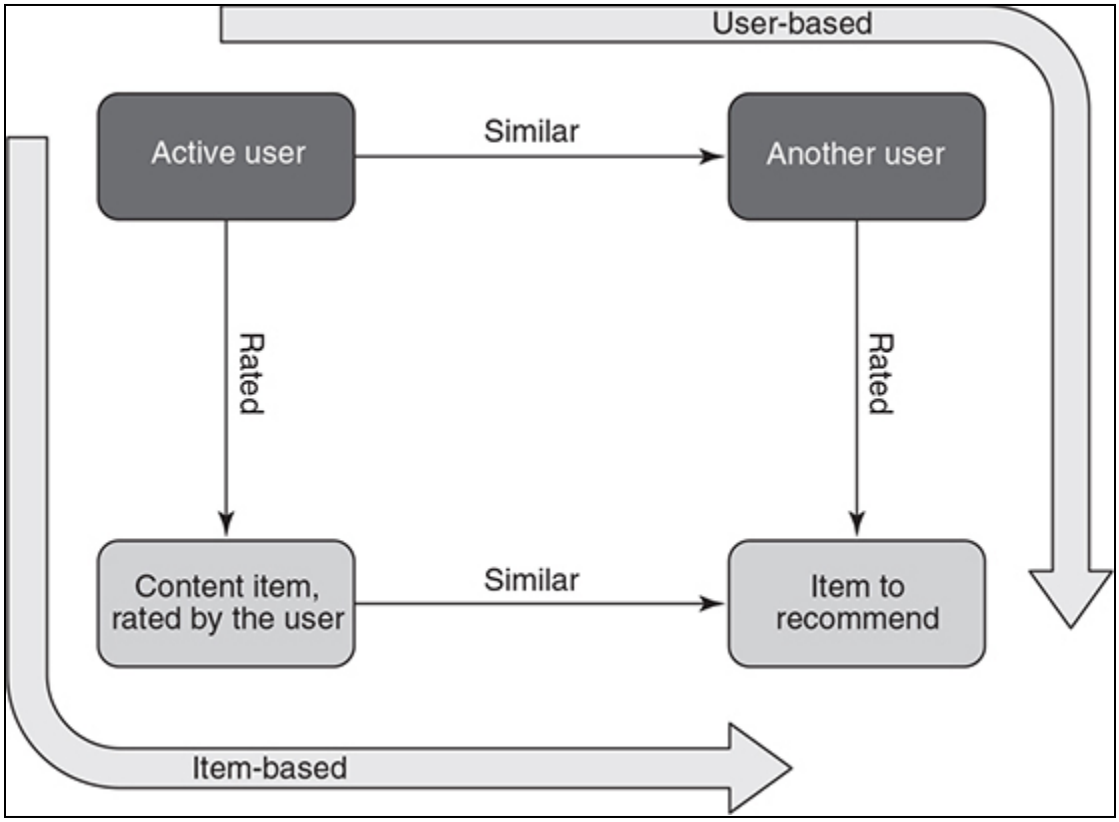
[8] https://livebook.manning.com/book/practical-recommender-systems/chapter-8/4
[9] https://developers.google.com/machine-learning/recommendation/collaborative/basics

Embeddings, or vector representations of users and items can be automatically learned in CF, eliminating the need for data gathering and preprocessing that is required in content-based filtering.

CF Neighbourhood Models

One of the first successful CF models was known as the neighbourhood model. Within the scope of neighbourhood models, both user- and item-oriented approaches exist. Should I explain matrix factorization and how this works? For both approaches, a user-oriented approach involves estimating unknown ratings or preference for an item by using recorded ratings made by like minded users. Conversely, an item-oriented approach predicts unknown ratings based on recorded ratings made by the same user on similar items.[10]

Figure 8.2. The two ways of performing neighbourhood-based filtering. One method uses similar users, while the other uses items similar to items the active user liked.



From: Practical Recommender Systems by Kim Falk

Regardless of which approach one chooses to use, the logic behind the model remains the same. A measure of similarity is used to calculate the affinity of two items or two users

[10] From presentation slides (https://docs.google.com/presentation/d/1Q5r--DGURmqowrukK2QfxEXOG82WiuiPfiBz00hqZPA/edit#slide=id.g117eb4d2b07_2_8), not sure if these were taken from somewhere?

(depending on an item- or user-based approach, respectively). This is done for the entire set of user or item neighbours, and the highest similarity user or item is used to generate a recommendation.

While this model is superior in many ways to that of content-based filtering, it still struggles with issues such as data sparsity and the cold start problem. The sparsity problem arises from the inevitable fact that given a large catalog of items, such as the Netflix movie catalog, it is unlikely that any given user has interacted with a large percentage of the total number of movies. The cold start problem comes from the issue of creating a recommendation for a user given that we have no prior data about them, for example a new user on Netflix.

CF Latent Factor Models

Another very successful collaborative filtering model is the latent factor model. The latent factor model aims to discover latent, or unknown features by projecting users and items onto a joint latent factor space.[11]

***Not sure if I should talk about feature space - the fact that the user and item embeddings are projected into a common d-dimension latent feature space is fine, but how much focus (if any) should be placed on the latent features themselves? In a lot of papers, they describe this is

Alternating Least Squares (ALS) in Recommender Systems with Implicit Data

In this section, the model of alternating least squares for generating recommendations is presented and analyzed. This model is derived from singular value decomposition (SVD) and matrix factorization, which form the basis of the algorithm.

Singular Value Decomposition (SVD) and Matrix Factorization

SVD is a powerful algorithm for data reduction and dimensionality reduction, and the basis for many machine learning models and data processing tools. It is a form of matrix factorization, crucial to the alternating least squares algorithm. SVD's usefulness comes from, in part, its ability to reduce data into its key features that are needed for analyzing, describing, and modeling data. It can be used to solve matrix systems of equations ($Ax = b$, linear systems of equations), for non-square A matrices. SVD is also often used as the basis for principal

---

[11] https://arxiv.org/abs/2012.03433#:~:text=Latent%20Factor%20Model%20(LFM)%20is,a%20joint%20latent%20factor%20space.

component analysis (PCA) - taking high dimensionality data, and trying to understand it in the context of its dominant patterns and correlations. [12]

At its core, SVD takes in high dimensionality data in matrix form, and distills it into key features and correlations in the data, which can be used to then interpret and model the data. It is also widely used in industry; it is used as the basis of the Google PageRank model, as well as in facial recognition algorithms, and of course, in recommender systems used by Google, Netflix, and Amazon, to build correlation patterns between items and users.[13]

The aim of SVD is the factorization of an *mxn* matrix, *M*, into three component matrices - the product of these three matrices is the original matrix, *M*.

$$M = U\Sigma V^T$$

Where *U* is an *mxr* matrix, Σ is an *rxr* matrix, and *V* is an *nxr* matrix.

While classical SVD is used in many recommender systems, matrix factorization is done in reverse in alternating least squares. In ALS, the original matrix *M* is being approximated by the product of two component matrices with the aim of imputing the missing values in the original rating matrix, and thus "predict" unknown ratings.

Matrix factorization in ALS at a high level is computed as follows. Suppose there exists a sparse matrix *R* of size *mxn*, where *m* (the rows of the matrix) represents users, and *n* (the columns of the matrix) represent the items. The matrix is very sparse as there are many missing ratings - not every user has rated every item. The entries of the matrix itself are ratings, either explicit or implicit. In other words, the entries of the matrix could be a rating from 1 to 5 (explicit) or the number of times a user may have watched a particular movie/item (implicit). Indexing for users is denoted as *u*, and *i* for items. An observed rating given to item *i* by user *u* is denoted as r_{ui}. Note that predicted a predicted rating is denoted as $\hat{r}_{ui}$ (r-hat).

The aim of this modified SVD or "reverse matrix factorization" is to impute the missing values in the matrix, which then make up all of the predicted ratings, from which we can pick the best to make a recommendation. Mathematically, the aim is to find two matrices, *U* and *V*, whose product approximates the original ratings matrix, *R*, according to the equation below.

$$R \approx UV^T$$

Where *U* is a matrix of dimension *mxd*, and *V* is a matrix of dimension *dxn*. The component matrices themselves are embeddings in *d*-space of the items and users, where *d* is a hyperparameter of the model (more on this below). Each user *u* is represented by a vector x_u of

---

[12] Data Driven Science and Engineering - Machine Learning, Dynamical Systems, and Control by Steven L. Brunton and J. Nathan Kutz (http://databookuw.com/)
[13] Ibid and https://www.youtube.com/watch?v=gXbThCXjZFM&ab_channel=SteveBrunton

length *d*, which make up the columns of the component matrix *U*. Similarly, each item *i* is represented by a vector y_i of length *d*, which make up the rows of the component matrix *V*.

It follows from the equation above that the dot product of any user and item vector will yield the predicted rating that that user *u* would give item *i*, or an approximation of an existing observed rating, r_{ui}.

$$\hat{r}_{ui} = x_u^T y_i$$

Predicted rating for a previously unknown rating.

$$r_{ui} \approx x_u^T y_i$$

Approximation of a previously known/observed rating.

Thus, the main goal of alternating least squares is to find the two component matrices, *U* and *V*, which best approximate the imputed matrix, *R*. The vectors representing users and items map serve to map both users and vectors into "the same latent feature space, where they can be directly compared".[14] The new imputed ratings in the approximated *R* matrix can be used as recommendations. The algorithm and loss function to find these matrices is presented below.

Confidence and Preference to Represent Implicit Data

Given explicit data, there is a clear cut meaning associated with a user-item interaction. A rating from 1 to 5, a form of explicit data in recommender systems, clearly tells a system if a user really likes or dislikes an item. Similarly, a missing rating is exactly that; an unknown interaction. There is a clear understanding of any explicit rating, or a lack thereof.

Given implicit data, this is not the case. For example, a recommender system is presented with an item-user interaction (ratings) matrix with songs (items) and number of song listens (ratings) that a user has for a particular song. It is possible to infer if a user likes a song - they may have listened to it many times, but implicit data is not symmetric. A missing observation (number of listens) could mean that the user does not like the song, but it is equally likely that they have not discovered the song. It is difficult to understand what a user does not like. Unlike with explicit data, both the missing and present observations/ratings are important in making a recommendation.

A numerical value of explicit data represents preference - given a rating from 1 to 5, it is clear what a user's preference for an item is. Implicit data on the other hand represents confidence - a higher number of interactions with an item gives a recommender system more confidence in the observation (i.e. that a user has a preference for an item). Implicit feedback is also noisy. It is data gathered from user behaviour, and leaves the recommender system to infer what it means. Following the same songs example, a number of plays may be due to a user making a playlist for a friend, or listening to songs they truly like.

[14] http://yifanhu.net/PUB/cf.pdf

To address the pitfalls of implicit data, the raw observations (r_{ui}) are transformed into two separate magnitudes; preference p_{ui}, and confidence c_{ui}. Preference is a value derived from binarizing the r_{ui} values as:

$$p_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \end{cases}$$

If a user has interacted with an item at all, there is some preference assumed (p_{ui} = 1). If a user has never interacted with an item, there is no preference (p_{ui} = 0).

Along with preference, a confidence score is calculated. Confidence grows with r_{ui}, as it follows that the more interactions with an item a user has, the higher the likelihood that they actually like it (have a preference). c_{ui} is measured as follows:

$$c_{ui} = 1 + \alpha r_{ui}$$

There is a constant factor of 1 applied to all r_{ui} to ensure a minimal confidence, and then confidence grows linearly with increasing r_{ui} values (controlled by the alpha factor - this is a hyperparameter of the implicit alternating least squares model).[15]

These are not the only ways to transform raw r_{ui} values into measures of confidence and preference. Both preference and confidence can be calculated with different functions, but the methodology presented above follows Hu et al.[16]

Alternating Least Squares (ALS) Algorithm

The goal of the alternating least squares algorithm is to take a sparse matrix of data (implicit or explicit) between users and items, and find two lower dimensionality matrices whose product best approximates the original matrix. Given these two matrices, which are embeddings of the users and items, the full imputed matrix can be estimated as their product, and used to generate recommendations.

The loss function for this model/algorithm is seen below:

$$\mathrm{L} = \sum_{r_{ui}} (r_{ui} - x_u^T y_i)^2 + \lambda(||x_u||^2 + ||y_i||^2)$$

For each known value of r_{ui}, the model is trying to find a vector *xu* (user vector) and vector *yi* (item vector) such that their product is as close to the true rating value as possible. These

[15] http://yifanhu.net/PUB/cf.pdf
[16] Ibid

vectors make up the component matrices whose product best approximates the original matrix but with imputed values.

The first term in the equation is the square of the loss between the known rating (r_{ui}), and the dot product of a user vector and item vector, whose dot product should be as close to the original rating as possible. The aim is to minimize this term (as close to 0 as possible).

The second term in the equation is required for regularization. λ is a hyperparameter of the model, and the whole regularization term is necessary in order to prevent the model from overfitting to the training data.

While this loss function would work for explicit data, it does not account for the aforementioned problems that are associated with implicit data. Given an observation matrix with implicit data, the loss function would be structured as follows:

$$\text{L} = \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda(\sum_u ||x_u||^2 + \sum_i ||y_i||^2)$$

The known r_{ui} values have been binarized and converted into preference values - p_{ui} is treated as “ground truth” and also the inner product $x_u^T y_i$. This is then scaled by the c_{ui} factor, which accounts for how much confidence we have in the ground truth p_{ui} values. Finally, given implicit data, this loss must be calculated over all user-item pairs, as opposed to explicit data, in which the loss needs to be calculated only for known r_{ui} values. This is because in implicit data, all observations are associated with a preference in order to avoid positive bias.

The “alternating” portion comes from how we solve for the user and item vectors that make up the component matrices. Since implicit data requires the loss to be calculated for all item-user pairs, and this is repeated until convergence, and given the fact that many datasets used in recommender systems are often made up of millions of observations, the optimization process becomes very costly ($O(m*n)$). The solution to this is to optimize the user and item vectors alternatingly.

By holding either the user or item vectors constant in each iteration of the algorithm, the alternating least squares task becomes solving a series of different loss functions.

Alternatingly, either the user vectors (x_u) and item vectors (y_i) are fixed in the cost function, while the non-fixed vectors are re-computed and optimized. In this way, with each iteration, the loss value should be lowered. Given that either the user vectors or item vectors are held constant, the loss function becomes quadratic and a global minimum can be calculated. Notice also that this function is convex, so there is assurance that there will be convergence at a global minimum (optimization).

We first differentiate the loss function with respect to the user vectors:

$$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u)$$

Interestingly, the solution for the user vectors is found by taking the derivative of the loss function with respect to the user vectors (item vectors are constants) and setting it to zero. Note however that this looks a lot like the normal equation, which is obtained by "setting equal to zero the partial derivatives of the sum of squares errors (least squares), which allows for the estimation of the parameters of a multiple linear regression".[17] This is exactly what is being done here, and this equation actually is the normal equation for a *regularized* and *implicit* least squares loss function.

The same can be done for the item vectors - the loss function is differentiated with respect to the item vectors:

$$y_i = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u)$$

Given these two equations for solving for the user and item vectors, the algorithm for implicit ALS involves alternating between computing these two equations, until convergence. Once a global minimum is reached, and the two component matrices (users and items) has been found, the imputed observation matrix can be calculated, and recommendations for any user-item combination can be made.

[17] https://link.springer.com/referenceworkentry/10.1007/978-0-387-32833-1_286