Matrix Factorization for Collaborative Filtering Recommender Systems

Jeremy Hintz

December 17, 2015

Introduction

Anyone who has recently gone shopping online has witnessed a wide variety and sometimes overwhelming volume of purchasing options. It is frankly unrealistic to think that a user could scroll through and view all these options to locate the items that he or she would like to purchase. For this reason, e-commerce companies must bring to the forefront those items which the individual users are most likely to buy, and so these companies use *recommender systems*, which both determine what items to suggest and a level of confidence in each of their suggestions. While recommender systems can be used to solve a wide range of problems [2], we illustrate the process of building recommender systems that attempt to pair a set of users with a corresponding set of items.

Recommender systems are generally divided into two main subcategories: content-based and collaborative filtering. In the content-based approach, users and items are characterized by certain properties in order to construct a kind of profile that describes each actor in the system. A quintessential example is a system where the items are homes and the users are home buyers. Content-based approaches would utilize information about the homes, perhaps such as price, geographic location (i.e. zipcode), and number of bedrooms/bathrooms to build a profile. Often, demographic information such as age, income, and number of children, is used to characterize users. A content-based recommender system would then work to associate properties of users with frequently co-occurring properties of items. For example, home buyers who are married with four children are more likely to buy homes with many bedrooms than buyers that are single with no children. From the perspective of most web services, the main problem with content-based recommender systems is that they require users to provide a large amount of information about themselves

Collaborative filtering is an alternate technique for determining which items to recommend that does not require users to input personal data. Instead, collaborative filtering utilizes the users' history of actions to match users with items, and users with similar histories are assumed to share key properties. This approach is usually more accurate for predicting user-item affinity than content-based methodologies [1] as it leverages all user histories to make better recommendations.

Key Ideas Behind Collaborative Filtering

The idea behind collaborative filtering is relatively intuitive. Consider Figure 1, which illustrates the case of three users who all have a set of items that they *like*. The specific definition of *like* varies by domain and application but could signify a previously purchased item, an item for which a five star rating was given, or any other indications of affinity that exist within the system context.

In Figure 1 users Alice, Bob, and Charlie each have some set of preferences indicated by letters a, b, ..., i. These users can share preferences. For instance, all users like item a. Similarly, Alice and Bob both like items a and f. Now let us imagine a new item j enters the system. Imagine that Bob immediately indicates an affinity for j. Then a logical question might be: if we recommend item j to Alice and Charlie, in which recommendation do we have a higher level of confidence?

Intuition tells us that a recommendation based on Bob's preferences is more likely to be liked by a user with similar "taste" to Bob. What indicates similar taste? While we can imagine many heuristics for guessing a user's taste, our current scenario lends itself to a simple metric: the larger the intersection of two users' preferences, the more similar their taste. Thus in this scenario, we

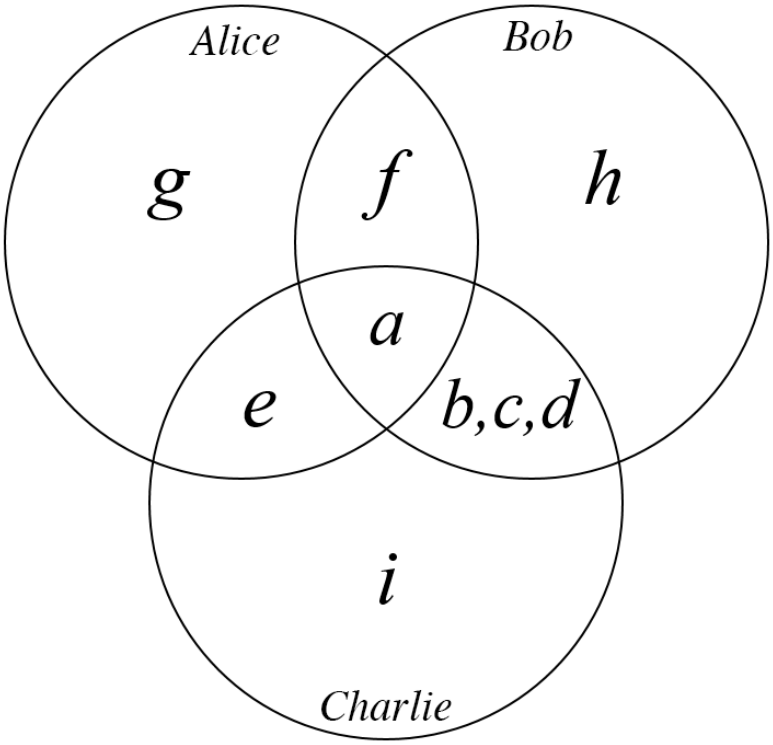

Figure 1: Common and unique preferences in a hypothetical system

would be more confident recommending j to Charlie, since Bob and Charlie like many of the same items.

Latent Factor Models

The content-based recommender systems we saw earlier paired items with user traits based on how often they occurred together. That approach made sense, but we encountered the difficulty of attaining information from users. For example, many children corresponding to a higher number of bedrooms made intuitive sense, but asking how many children someone has, along with many other questions necessary for building a complete user profile, is intrusive. Web companies thus mostly rely on the data they receive through interactions between their service and their users.

Models built on said data are called latent factor models. A *latent factor* is one of the properties in a user profile minus the explicit knowledge of that property. Latent factors are the underlying force behind the decisions that users make. For instance, even if we don't explicitly know that a home buyer has children, we can infer that they may since they viewed homes with many bedrooms. Thus if two home buyers show interest in multiple homes with many bedrooms, then it would be reasonable to show the buyers the homes that they themselves had not yet viewed but their counterpart had viewed. Furthermore, latent factor models are very good at identifying traits that are difficult to specifically include as properties of the item. For instance, users with children may also be interested in homes in cul-de-sacs or homes near good schools. Even if our service did not contain this information, a good latent factor model would coerce it from the data in an unsupervised fashion.

Latent factors, just like known features, can be discrete/categorical or continuous values. They are kept in a list or array of values, as discussed further in the next section. Collaborative filtering models that use latent factors tend toward better accuracy as the number of latent factors increases [1].

Matrix Factorization

While the Venn diagram in Figure 1 helps us understand the idea behind collaborative filtering, collaborative filtering algorithms actually represent the data in matrix form.

Transforming our data into matrix form is done as follows: let r be a matrix with the rows representing users and columns representing items. Let us fill in all $[u, i]$ cells of r with a 1 if user u likes i and 0 otherwise. Similarly, we may be able to measure how much u likes i and can instead use some continuous value for entry r_{ui}. In an explicit feedback system, this could be done through a rating system. For instance, r_{ui} could be filled with numbers 1-5 representing a 5-star scale. Similarly, in an implicit feedback setting, our confidence in an affinity pairing could, for example, increase according to the rules of some separately trained logistic regression.

We will formulate our model for achieving the most accurate recommendations by, as with most machine learning models, optimizing some cost function. While this cost function and the manner in which we locate its maximums and minimums changes with the variation of matrix factorization we choose to employ, the general technique for matrix factorization is as follows: consider that our $m \times n$ matrix r can be factored into two matrices x and y of dimension $f \times m$ and $f \times n$, respectively. When using the concept of latent factors, we can let f be the number of latent factors prescribed for our model, recalling that a higher number of latent factors yields more accuracy but comes at the expense of computational efficiency (by the properties of matrix factorization itself).

Now let's assume that we took all the columns of x, the user vectors, and populated them randomly. Each x_u is a column vector called the latent factor vector for user u. Similarly y_i is the latent factor vector for item i. Then multiplying x^T and y would give us a matrix that has the same dimensions as r but does not equal r. In order to be able to say that we have found latent factor vectors that coincide with our collected data, we want to minimize the sum of cells of r - $x^T y$. This is to say we want the multiplication of our decomposed matrix to approximate as closely as possible our known data from r. Thus the general form for matrix factorization is as follows:

$$\min \sum_{u,i} (r_{ui} - x_u^T y_i)^2 + \lambda(||x_u||^2 + ||y_i||^2)$$

Here, the last term is a regularization term. λ is a regularization hyperparameter that must be tweaked to minimzie overfitting. There are different varieties of matrix factorization that stem from the above equation. They differ primarily in how they go about the process of minimizing r - $x^T y$. A few notable examples are discussed below.

Implicit Matrix Factorization

Implicit matrix factorization (IMF), as described by Hu et al [1], is widely used in collaborative filtering problems. The methodology is so widely accepted that it comes standard in many popular machine learning libraries such as MLLib for Apache Spark. The authors' algorithm has the following cost function:

$$\min \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda(||x_u||^2 + ||y_i||^2)$$

Note the subtle differences between the authors' technique compared to the more general form. Here c_{ui} is called the confidence and is given by $c_{ui} = 1 + \alpha r_{ui}$. The confidence is used in order to give some minimal confidence for every user-item affinity pairing (i.e. 1 in this case), which can be subsequently incremented based on the number and/or significance of interactions between that user and item. Here α is a learning rate parameter that can be tweaked in order for the minimization to converge. The matrix p_{ui} is populated from r_{ui}. Anywhere where there is a non-zero value in r, p is set to 1. Entries for which r is 0 remain as such in p. Finally $x_u^T y_i$ is the product of the user vectors and the item vectors. So clearly, just as before we wish to minimize the difference between our known data and data we have predicted for our user and item latent factors, scaled by our confidence in the predictions. Because we alternate holding the user vectors and item vectors constant, one of them being fixed makes our cost function above quadratic. This readily suggests that an alternating least squares methodology will make the most sense. Indeed, that is what the authors of [1] use.

We terminate the protocol after a specified number of iterations on our cost function or once we have reached some acceptable level of convergence. At this point, we have an $m \times n$ matrix that represents predictions for the affinity of given user-item pairs. Typically for most applications, we will want to take say the k highest affinity scores and say these are a users "top items" and recommend these items.

Logistic Matrix Factorization

Another methodology for matrix factorization that is currently trending due to its use at Spotify is Logistic Matrix Factorization (LMF). It is similar to IMF, but uses probabilistic approach to formulating the prediction for an affinity value. Concretely, whereas the IMF and other forms of collaborative filtering look to predict whether a user will engage with an item and assigns a discrete guess of 1 for yes and 0 for no, the LMF looks to predict a continuous value for the probability that a user will engage with an item. The new cost function then looks like this:

$$\min \sum_{u,i} \alpha r_{ui}(x_u y_i^T + \beta_u + \beta_i) - (1 + \alpha r_{ui}) log(1 + exp(x_u y_i^T + \beta_u + \beta_i) - \lambda(||x_u||^2 + ||y_i||^2)$$

While the new LMF cost function may seem more complicated than the one used for IMF, the principle is the same. We simply have modulated all of the interactions between our known matrix and latent factor vectors using a logit function. Because of this, we can no longer utilize a simple alternating least squares minimization and must opt instead for a slightly more complicated (but hopefully just as intuitive) alternating gradient descent approach. In order to do this, we use the fact that the partial derivatives of our cost function, f, are as follows:

$$\frac{\partial}{\partial x_u} = \sum_i \alpha r_{ui} y_i - \frac{y_i(1+\alpha r_{ui}) exp(x_u y_i^T + \beta_u + \beta_i)}{1 + exp(x_u y_i^T + \beta_u + \beta_i)} - \lambda x_u$$

$$\frac{\partial}{\partial x_i} = \sum_u \alpha r_{ui} x_u^T - \frac{x_u^T(1+\alpha r_{ui}) exp(x_u y_i^T + \beta_u + \beta_i)}{1 + exp(x_u y_i^T + \beta_u + \beta_i)} - \lambda y_i$$

Thus our gradient descent simply freezes either the user or item vectors and attempts to take a step toward convergence using the above partial derivatives.

Online Matrix Factorization

The last flavor of matrix factorization that we will consider is that of Online Matrix Factorization (OMF) as described in [4]. In machine learning, the term online learning refers to an algorithm that can run on streaming data as it comes in, as opposed to training on batch data and periodically retraining on a larger and larger batch as new data comes in. This is extremely useful for practical applications as the task of retraining a model and having the results of new model reflected in the output of whatever recommender service is serving up the recommendations is certainly non-trivial for most applications of meaningful scale. Another tremendous benefit of online learning is that our model is more up to date with recent data, since there is no lag time between retraining the model.

Before we give the cost function for OMF, we must note that one significant modification that the authors of [4] make is to use a low-rank approximation of the known matrix. Because of the sparsity of the matrix, it is reasonable to do so. The authors include discussion of how the approximation affects the quality of the recommendations made, but claim that the slight loss of precision is greatly outweighed by the gain in efficiency.

The cost function for OMF is as follows:

$$\min (\sum_l x_u y_i^T - r_{ui})^2 + \lambda \sum_i \sum_j \alpha \beta_i^T \beta_j$$

Just as before, we are minimizing the difference between the known values for user-item interaction and the predicted values. The regularization must be changed, however, to deal with the low-rank approximation done to our matrices. The process of minimization occurs through alternating gradient descent. While the full protocol for the technique of Online Matrix Factorization can be found in [4], but it is worth noting that online gradient descent is more computationally demanding than its batch counterpart, and thus even with the approximation of our matrix, the time and space efficiency of OMF must be managed. The algorithm is parallelizable by partitioning the matrix.

Model Evaluation

As with any machine learning model, we need some way to evaluate the recommendations we receive back from our matrix factorization models. Many metrics exist, some of which are likely familiar to many other applications such as the Mean Squared Error (MSE), which is simply the average error for each recommendation. A metric that is relatively unique to matrix factorization is Mean Percentile Rank (MPR). Mean Percentile Rank uses the notion that our predicted recommendations should intuitively have as "top picks" the items for which the user actually indicated affinity with the model's new guesses added into the mix. MPR takes each item from r_{ui}, calculates the percentile of that item within the list of all items, and then takes the average percentile. If we assigned recommendations at random, the expected value of the MPR would be 50%. Similarly simply recommending popular items to every user would typically yield an MPR of somewhere around 20%. Thus, when evaluating our model, we wish to see concrete improvement with an MPR lower than 20%, since if we can't get lower than that, then we are better off recommending items that are broadly popular or trending.

Conclusion

Recommender systems are seen as an essential part of the business model of many major tech companies. The personalization of product recommendations and advertisements has become paramount to the success of web services and online retailers. Within recommender systems, collaborative filtering using matrix factorization is nearly ubiquitous because of its parallelizability and relative computational efficincy. Collaborative filtering with matrix factorization is progressing to allow for quicker turn around time between new data coming in and a new model running on the personalization service.

References

[1] Hu, Yifan, Yehuda Koren, and Chris Volinsky. "Collaborative filtering for implicit feedback datasets.", *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on.* IEEE, 2008.

[2] Koren, Yehuda, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." *Computer* 8 (2009): 30-37.

[3] Johnson, Christopher C. "Logistic Matrix Factorization for Implicit Feedback Data." 2012.

[4] Ling, Guang, et al. "Online learning for collaborative filtering." Neural Networks (IJCNN), *The 2012 International Joint Conference on.* IEEE, 2012.