A Survey of Techniques in Recommender System

A Final Project Report for CS 590OP

Long Le
University of Massachusetts Amherst

Steven Qiu
University of Massachusetts Amherst

Fall 2019

Abstract

This is a final project report for *CS 590OP: Numerical Optimization* at UMass Amherst with Prof. *Andrew Lan* . In this report, we provide an exposition and some experiments for three methods used in recommender systems, including alternating least square, locality sensitive hashing, and stochastic gradient descent. These methods are introduced to us through CS 589 taught here at UMass.

***K*eywords** Collaborative Filtering · Matrix Factorization · Locality Sensitive Hashing · Stochastic Gradient Descent

1 Background

Recommender system is a mechanism for predicting an user's preference for a product, called item, based on the information that we collect on this user and other similar users. Such systems has tremendous practical values and are vital to the business of many corporates such as Amazon and Netflix. There are roughly two types of filtering: content-based and collaborative filtering. In content-based filtering, some sort of meta-data or profile about the users is built to predict their behavior using user-information such as their age or ethnicity. Collaborative filtering (CF), on the other hand, only relies on the user's past behavior. In this project, we focus on CF approaches with latent factor and neighborhood methods.

2 Optimization Formulation and Algorithms

The input to our recommender system is a matrix $Y \in \mathbb{R}^{m \times n}$ consisting of rating from n users and m items. Some of the entries in the matrix might be missing, presenting missing ratings from items that a user has not rated. The problem is fairly open-ended: to obtain a good prediction for those missing ratings. This problem will be formulated into an optimization problem as follows.

2.1 Linear Dimensionality Reduction

Often, Y is a very large matrix as we can have many users and many items. We would like to find a lower-dimensional representation of Y. The idea here is that to both users and items can be thought of as points living on a low-dimensional space. For example, users and movies can live in the genre spaces. The number of movie genre such as action, romance is much smaller than the number of movies and users.

If we can manage to find this low-dimensional embedding, we can project both users and items down to this space, and calculate the proximity between items and users. A user who is close in space to a movie is predicted to rate that items highly.

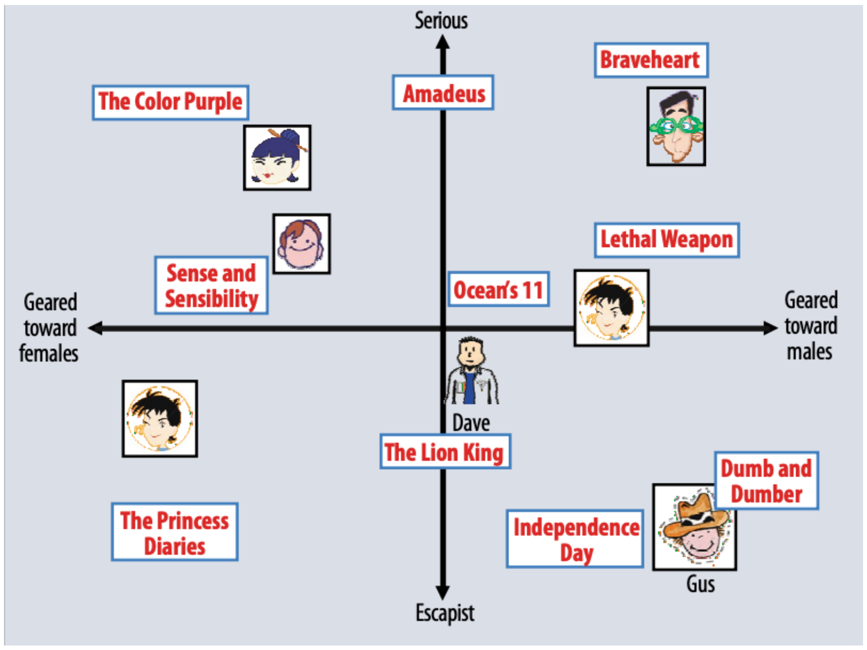


Figure 1: Low-dimensional Embedding (Source: Yehuda Koren, Robert Bell, Chris Volinsky from Yahoo Research)

Formally, we would like to find a k-dimensional embedding consisting of $X \in \mathbb{R}^{m\times k}$ and $\theta \in \mathbb{R}^{k\times n}$ such that

$$\min_{X,\theta} \|Y - X\theta\|_F$$

where $\|.\|_F$ is the Frobenius norm. We think of X and θ as the users and items embeddings respectively. k is known as the number of latent factors. To solve this optimization problem, we employ a technique known as alternating least squares (ALS). ALS is a maximization-maximization procedure. We first fix θ and optimize X. With fixed θ, the optimal X is simply the solution to an ordinary least square problem. Then we alternate with fixing X and optimize X. This technique is very similar to other techniques such as expectation-maximization.

```
Initialize $X$ and $\theta$ randomly
while in iteration do
    $X^T = (\theta\theta^T)^{-1}\theta Y$
    $\theta = (X^T X)^{-1} X^T Y$
end
return $X, \theta$.
```

Algorithm 1: ALS

For simplicity, we fill missing entries in Y with zeros. Then, we set ALS to run for T iterations and compute $\hat{Y} = X\theta$ as an approximate for Y. $\hat{Y}$ also provides predictive ratings for missing entries.

From a theoretical perspective, $\hat{Y}$ is the optimal solution for the Frobenius norm in linear reduction. However, $\hat{Y}$ is not unique. We can obtain a unique solution over the same linear subspace by using rank-k singular value decomposition (SVD). This means solving the following optimization problem

$$\min_{U,S,V} \|Y - USV^T\|_F$$

where $S \in \mathbb{R}^{k\times k}$ diagonal, $U \in \mathbb{R}^{n\times k}, V^T \in \mathbb{R}^{k\times m}$ orthogonal. It can also be shown that SVD and principal component analysis (PCA) compute the same linear subspace. But PCA maximizes the retained variance in the subspace. Thus, SVD also enjoys the same property. Note that in our project, we only uses ALS since uniqueness is not our concern. The discussion here is simply intended to provide some theoretical properties of our solution $\hat{Y}$.

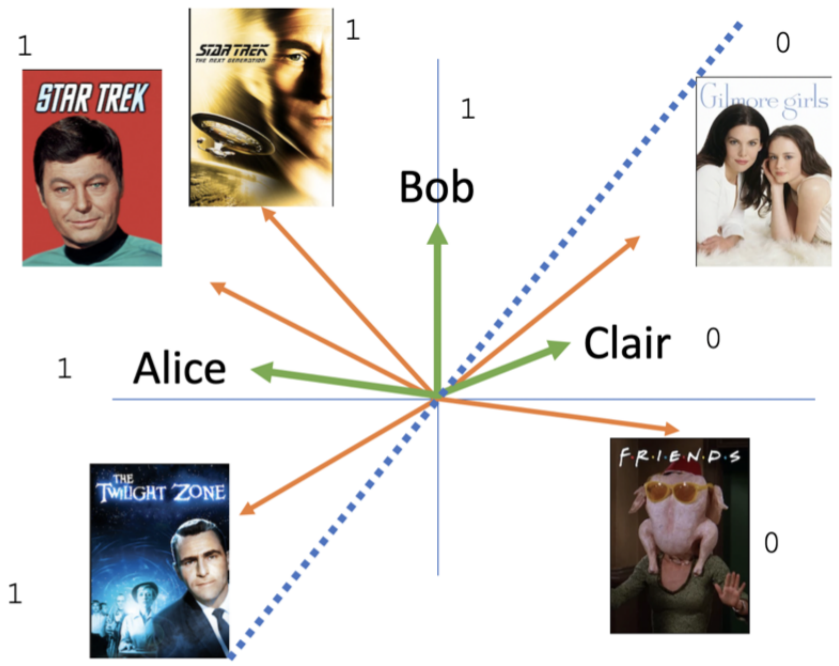


Figure 2: Random direction to partition the space (Source: CS 589, Rob Hall)

2.2 Locality Sensitive Hashing

Once we compute the user and item embedding, we are still faced with a challenge: prediction. For example, to recommend a product to an user, we have to compute m dot products with that user and every products in our database using the pre-computed embedding. For very large m, this would be very slow in real-time practical system.

This is similar to exhaustively searching through every point in the embedding space to find some k-nearest neighbour to the user and recommend those k products to them. An approximate solution to this problem is Locality Sensitive Hashing (LSH). The idea is to construct a hash function to put similar users and items in the same hash bucket. Then, given a user, we look into their hash bucket and find k nearest items from that bucket only instead of considering all items in our database. Note that this is different from usual hash function construction, where we try to avoid collision as much as possible for fast query. Here, we intentionally create collision whenever an item is similar to a user.

LSH does this by iteratively generate a hash bit for each point (items/users) in the embedding space. To generate a hash bit, we pick a random line through the space, which partitions the space in half. We assign 1 to all points on one half-space and 0 to the others. This is shown in figure 2.

We repeat this process until we have generate b-bit encoding for each point, where b is a hyper-parameter. In details, we create an extra dimension to augment the embedding space such that each item has the same fixed norm. To generate the i^{th} bit for the point p, we generate a random multivariate Gaussian vector z. The i^{th} hash code for p is simply which side of the space p lies on.

$$H_i(p) = sign(p^T z).$$

For motivation, we provide a crude explanation for 2-D embedding. The dot product between user u and item v is $u^T v = \|u\| \|v\| \cos\theta$ where θ is the angle between the two vectors, measuring their similarity. We can assume that items have been normalized to have norm 1 so that $u^T v = \|u\| \cos\theta$. Since the random direction z is generated so that the angle of z with the x-axis is uniform, we have

$$p(H_i(u) \neq H_i(v)) = \frac{\theta}{180}$$

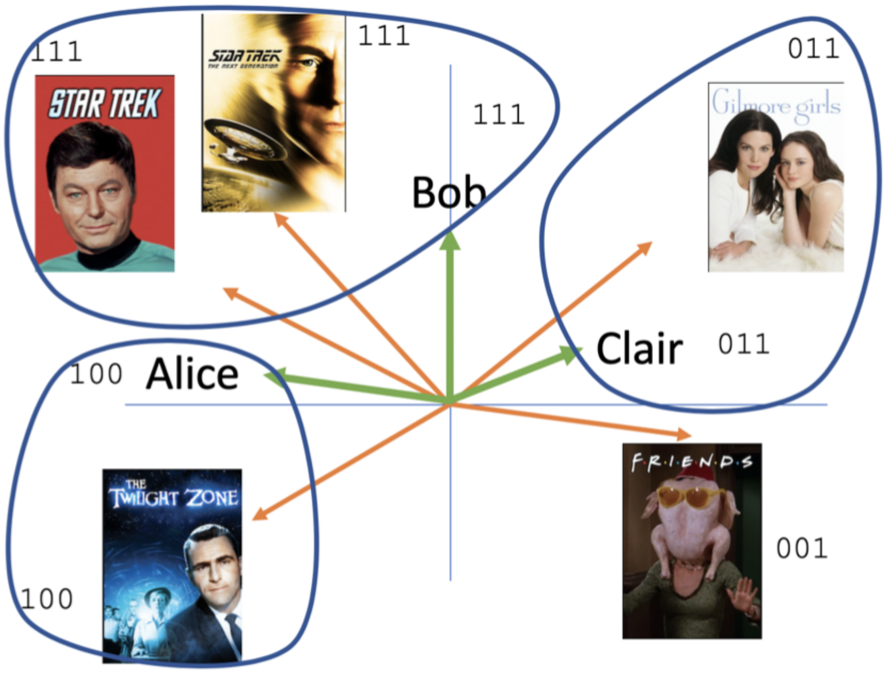


Figure 3: We only recommend items in the same bucket as the user (Source: CS 589, Rob Hall)

for each bit. Since each bit is generated independently, we have

$$p(H(u) = H(v)) = \left(1 - \frac{\theta}{180}\right)^b.$$

For large b and large θ, this probability vanishes so disimilar users and items are not likely to be in the same bucket.

2.3 Stochastic Gradient Descent

We may utilize knowledge from statistical learning to treat this problem as a linear model. After centering the ratings by subtracting "3" from the entire ratings matrix, we establish a low-rank reconstruction of the entries in Y by assuming that the users for each row in $Y_{\text{full}} : \mathbb{R}^{M\times N}$ has some hidden sets of "traits" $X : \mathbb{R}^{M\times k}$, $k < N$ centered around zero with a linear relationship $X = Y_{\text{full}}\beta$, $\beta : \mathbb{R}^{N\times k}$. We may then obtain an approximation of X by calculating the matrix product for the incomplete matrix Y to get $\hat{X} = Y_{\text{sparse}}\beta$. To get our result back, we may then calculate $\hat{Y} = \hat{X}\beta^{-1} = Y_{\text{sparse}}\beta\beta^{-1}$.

With this inspiration from statistical learning, we can formulate this problem a bit differently by approximating Y_{full} using $\bar{Y} = Y_{\text{sparse}}pq$ with $p : \mathbb{R}^{N\times k}$ and $q : \mathbb{R}^{k\times N}$. In this way, we have eliminated the requirement that $qp = \mathbb{I}$, which adds some flexibility for bias and other irregularities in the model. This, in essence, is attempting to transform the ratings into a latent "traits" space with p by assuming a neutral position on movies not rated, then transform the "traits" back to the ratings to predict a rating that users would give to movies.

We would then find the optimal projections p, q by performing a numerical optimization algorithm from a small perturbed initialization of p, q to find the best approximation by minimizing the sum of squared error from the existing entries in Y_{sparse} and magnitudes of the maximum singular values of p, q. To take in the many advantages offered by stochastic gradient descent, we may formulate the problem into finding a low rank prediction matrix and perform the alternating least squares in batches. Let $\Omega = \{c = (i, j)\}$ denote the position in which the original matrix Y contains ratings, we have:

$$\underset{p,q}{minimize} \sum_{c\in\Omega} [(Ypq)_c - Y_c]^2 + \lambda\|p\|^2 + \lambda\|q\|^2$$

In this formulation, we attempt to create a low rank matrix by the product of $p : \mathbb{R}^{N\times k}$ and $q : \mathbb{R}^{k\times N}$ to find an approximation $\hat{Y} = Ypq$ and then perform alternating least squares on batches. The least squares within each iteration is found using the line search method using the gradient calculation as shown below:

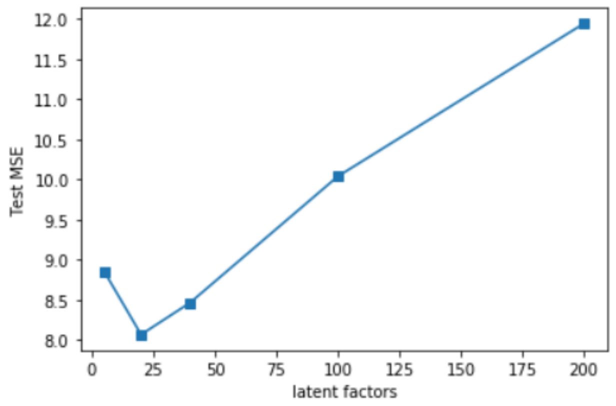
$$\frac{\partial L}{\partial p} = 2\sum_{c\in\Omega}\left[((Ypq)_c - Y_c)(Y_i({q^T}_j)^T)\right] + 2\lambda p$$

$$\frac{\partial L}{\partial q} = 2\sum_{c\in\Omega}\left[((Ypq)_c - Y_c)(Yp)_i\right] + 2\lambda q$$

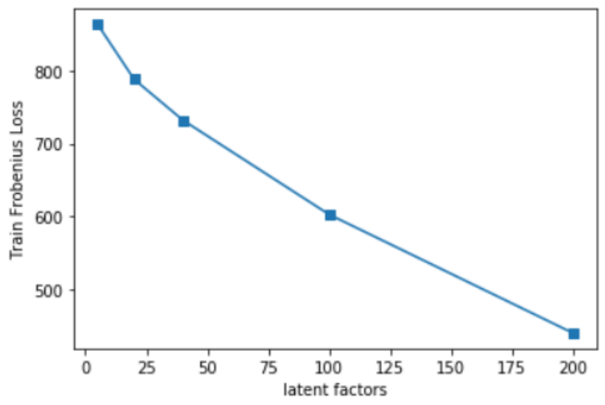
The term $(Yp)_i$ in the gradient calculation for q should correspond to only row i in the gradient, but I have no idea how to express that.

3 Result and Future Work

We use a subset of the MovieLens Dataset consisting of ratings from 943 users for 1682 movies. We pick 10 ratings out of potential 1682 ratings from each user to use as hold-out testing data. The number of latent factors (the rank of the approximation) is varied and the result for training and testing set is shown in figure 4.



(a) Number of latent factors vs Test MSE



(b) Number of latent factors vs Train Loss

Figure 4: ALS Result

For the setup in SGD, we performed 10,000 iterations of stochastic gradient descent using a rank of $k = 10$ and were able to reduce the sum of squared errors from an initial 154778.16 down to a final 47525.667.

For future work, we would like to experiment on large dataset. For matrix factorization, we also would like to implement other methods such as SVD instead of ALS and compare the performance. For LSH, we would like to research the recall metrics and formulate our hashing problem such that we can have methodology to evaluate our work.

4 Code

Section 2.1: Please see here for code. Section 2.3: Please checkout the GitHub repository and download the appropriate MovieNet 100k dataset for code.

References

https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf
https://movielens.org/
https://www.youtube.com/watch?v=NtAKQIrIU7w
https://towardsdatascience.com/creating-a-hybrid-content-collaborative-movie-recommender-using-deep-
https://colab.research.google.com/github/google/eng-edu/blob/master/
ml/recommendation-systems/recommendation-systems.ipynb?utm_source=
ss-recommendation-systems&utm_campaign=colab-external&utm_medium=referral&utm_content=
recommendation-systems#scrollTo=fF6dMP1To4uH
https://blog.codecentric.de/en/2019/07/recommender-system-movie-lens-dataset/
https://towardsdatascience.com/how-to-build-a-simple-recommender-system-in-python-375093c3fb7d
https://sigopt.com/blog/bayesian-optimization-for-collaborative-filtering-with-mllib/
https://sandipanweb.wordpress.com/2016/07/02/using-low-rank-matrix-factorization-for-collaborative-f
https://blog.insightdatascience.com/explicit-matrix-factorization-als-sgd-and-all-that-jazz-b00e4d9b