STA832 **April 20, 2017**

Ilan Man

Matrix Factorization for Recommender Systems

1 Introduction

In today's world of many options and choices, people often seek ways to reduce their decision set. Being accurately recommended a product, movie or song can simplify our decision making. Smart recommendations can also introduce us to products that we're likely to enjoy and perhaps wouldn't have considered otherwise. As one can imagine, companies routinely invest a lot of resources into building accurate recommendation engines. For many companies, recommendations form part of the product itself. This is explicit in firms such as Netflix or Amazon ("recommended for you"), or implicit in firms such as Facebook or LinkedIn ("news feed"). This paper will explore one type of recommendation engine using latent factor modeling via matrix factorization[1]. We apply this model to a MovieLens dataset and evaluate individual user recommendations.

2 Overview of Recommender Systems

Broadly speaking, recommender systems are based on one of two strategies:

- *Content Filtering*: This approach is based on creating a profile for each user or product. Movie profiles might be described using genre, leading actors or plot line. Then the recommendation system associates users with movie profiles. For example, a user might like action movies starring Jean Claude Van Damme, in which he competes in a martial arts tournament. For this user, the recommendation system will recommend movies that fit this profile, of which there are plenty.

- *Collaborative Filtering*: This approach relies on past user behavior rather than a user profile. Furthermore, it accounts for interdependences among items to identify new user-item associations. So even if a user has never used a certain product, their past behavior may indicate that they would enjoy it.

One advantage of collaborative filtering is that it's data-agnostic - all the algorithm cares about is user behavior, without having to understand the product itself. Collaborative filtering is based on the assumption that people with similar behaviors in the past will continue to have similar behaviors in the future. The two primary types of collaborative filtering models are the neighborhood methods and latent factor models. The focus on this paper will be on collaborative filtering using latent factor models.

Latent factor models are based on the notion that there exist certain factors which lead to the observed data. We say these are latent because we don't observe the factors - they are somehow hidden in the observed data and the task is to learn what those factors are. Some of these factors may be obvious, such as action movie vs comedy, and some are less well defined such as number of "dad jokes" or diversity of cast. In practice, we may not know what the actual factors represent, but in theory they represent what it is about a movie or product that will cause a user to rate that movie or product high or low.

2.1 Data

We'll focus the rest of the paper on movie recommendations. We are given a dataset with users, movies, and user ratings for each movie. Ratings range from 1 (strongly dislike) to 5 (strongly like), in 0.5 increments. Note that not all users have rated all movies. Latent factor models allow us to infer ratings for movies that a given user has not rated/seen. The recommendations are movies which have a high predicted rating.

Before discussing how we find these latent factors, a word about the data structure. For our purposes, we require that the data be stored as a matrix. The rows of the data matrix will represent each individual user and the columns each movie. The entries of the matrix will be the rating that a user gives to a movie. These ratings are gathered explicitly beforehand by some mechanism such as a survey or questionnaire. As mentioned above, many users will not have watched many of the possible movies. Their ratings are represented as 0. This results in a very sparse matrix.

We assume that a rating of 0 indicates that a user did not watch a movie, rather than the user refused to rate the movie or that the rating is missing. In general, missingness can be modeled as well. For example, imagine a user having rated all kinds of 80's action movies, but having never rated the classics. This suggests some sort of non-random relationship. For the purposes of this paper, however, we will treat all missing data as missing completely at random. The goal is then to infer what a user would have rated a movie, and if that rating is high, make the recommendation.

3 Latent Factor Model

3.1 Matrix Factorization Methods

As alluded to above, we will focus on data that is matrix structured. Some of the most successful realizations of latent factor models are based on matrix factorization. In general, a matrix factorization is simply a decomposition of a large data matrix into multiple, smaller matrices. Our matrix factorization model maps a data matrix of rank n to a joint user-movie latent factor space of rank $k < n$, such that user-movie interactions are modeled as inner products in that space. Concretely, define $\mathbf{u}_i$ as the user vector for user i and $\mathbf{v}_j$ as the movie vector for movie j. Then the joint factor space (each user-movie pair), which attempts to estimate the true rating, r_{ij} is given by:

$$\hat{r}_{ij} = \mathbf{u}_i^T \mathbf{v}_j = \sum_{m=1}^{k} u_{im} \cdot v_{mj}$$

Where $\mathbf{u}, \mathbf{v} \in \mathbb{R}^k$, i.e. k-dimensional column vectors. The challenge is to compute the mapping of each movie and user to factor vector, $\mathbf{u}, \mathbf{v}$.

High correspondence between movie and user factors leads to a higher rating and hence a recommendation. Matrix factorization methods have become popular in recent years by combining good scalability with predictive accuracy. In addition, they offer a lot of flexibility for modeling various real-life situations.

Arguably the most popular matrix factorization technique is the Singular Value Decomposition (SVD). However, one cannot simply apply the SVD to the ratings matrix for a few reasons. First, factoring the user-movie ratings matrix may result in an undefined SVD because of the sparsity of ratings. Secondly, only addressing non-empty entries is highly prone to overfitting since we'd be ignoring all the information in the missing data. One can perform missing data imputation as alluded to above, however, for the sake of keeping computational complexity to a minimum, we instead will learn the factors by performing squared-error regularization (i.e. ℓ_2 penalty) on the set of known ratings as follows:

$$\min_{u,v} \sum_{i,j \in K} (r_{ij} - \mathbf{u}_i^T \mathbf{v}_j)^2 + \lambda \cdot (||\mathbf{u}_i||^2 + ||\mathbf{v}_j||^2)$$

Here K is the set of known user-movie rating pairs for which r_{ij} is known. We denote this as the training set. Using cross-validation, we can solve for the regularization parameter, λ. However, it's also common to use small values on the order of $[0.01 - 0.1]$.

3.2 Learning Algorithm

There are a few ways to solve the above equation, including Alternating Least Squares (ALS) and Stochastic Gradient Descent (SGD). This paper will focus on the latter.

SGD involves looping over every rating in the training set and computing the squared error from the model prediction:

$$e_{ij}^2 = (r_{ij} - \mathbf{u}_i^T \mathbf{v}_j)^2 + \lambda \cdot (||\mathbf{u}_i||^2 + ||\mathbf{v}_j||^2)$$

Here we consider the squared error because the estimated rating can be either higher or lower than the real rating. We note that squared loss functions penalize mistakes very much. For example, if the true rating is 1 and the model predicts 5, the loss function is $(1-5)^2 = 16$. While squared loss makes computational and theoretical sense, there are other loss functions in the literature. For example, predicting a 2 when the true rating is 3, is perceived as a negative rating and the movie won't be recommended. However, predicting a 4 when the true rating is 3, is perceived as a positive rating and the movie should be recommended. In both cases, the squared error is the same (i.e. $(2-3)^2 = (4-3)^2$) but the practical outcome is very different. Hence ratings in this case are not truly uniform. Another example: when making recommendations, we care more about our recommendations being correct (correctly predicting a 5) compared to missing out on recommendations (incorrectly predicting ¡ 5). That is, we want to minimize our false positives more than we want to minimize our false negatives. This skewed decision function is another example of a non-squared error loss function.

To minimize the squared error, we have to know in which direction we need to modify the values of $\mathbf{u}_i$ and $\mathbf{v}_j$ and take a step in that direction. To do this, we must calculate the gradient at the current values; therefore we differentiate the above equation with respect to these two variables separately. Having obtained the gradient, we can now formulate the update rules for both $\mathbf{u}_i$ and $\mathbf{v}_j$:

$$\mathbf{u}_i' = \mathbf{u}_i + \alpha \cdot \frac{\partial}{\partial \mathbf{u}_i} \cdot e_{ij}^2 \propto \mathbf{u}_i + \alpha \cdot (e_{ij} \cdot \mathbf{u}_i - \lambda \cdot \mathbf{v}_j)$$

$$\mathbf{v}_j' = \mathbf{v}_j + \alpha \cdot \frac{\partial}{\partial \mathbf{v}_j} \cdot e_{ij}^2 \propto \mathbf{v}_j + \alpha \cdot (e_{ij} \cdot \mathbf{v}_j - \lambda \cdot \mathbf{u}_i)$$

Here, α is a constant whose value determines the step size in the direction of the gradient. Usually we will choose a small value for α. This is because if we make too large a step towards the minimum we may run into the risk of missing the minimum and end up oscillating around it. Of course, if α is chosen too small, then it will take longer for the algorithm to converge.

Using the above update rules, we can then iteratively perform the operation until the error converges to its minimum or reaches some threshold of convergence.

$$Error = \sum_{(u_i,v_j)\in K} e_{ij}^2 = \sum_{(u_i,v_j)\in K} (r_{ij} - \mathbf{u}_i^T \mathbf{v}_j)^2 + \lambda \cdot (||\mathbf{u}_i||^2 + ||\mathbf{v}_j||^2)$$

3.3 Adding Biases: an AMMI model

The task above attempts to model a true user-movie rating by modeling the interaction between users and movies. This assumes that all of the variation in ratings is due to this interaction. In reality, however, it's clear that many users rate movies according to their own idiosyncracies. For example, some users may tend to rate movies more highly, on average, than other users. Or perhaps some users enjoy comedies to a greater degree more than action movies compared to other users.

Likewise, some movies may tend to get higher ratings, all else equal, than other movies. For example, Hollywood blockbusters are likely to get higher average ratings because of heavy advertising and positive media compared to less well-known movies.

Note that this is known as an Additive Main Effects and Multiplicative Interaction (AMMI) model. In our case the user bias is similar to the row effect and movie bias is the column effect. To account for these biases, we introduce an overall average rating, and user specific, and movie specific bias terms, denoted μ, b_i and b_j respectively. Therefore we can estimate a user-movie rating by including a first-order bias approximation as follows:

$$\hat{r}_{ij} = \mu + b_i + b_j + \mathbf{u}_i^T \mathbf{v}_j$$

And the task becomes:

$$\min_{u,v,b} \sum_{i,j\in K} (r_{ij} - \mu - b_i - b_j - \mathbf{u}_i^T \mathbf{v}_j)^2 + \lambda \cdot (||\mathbf{u}_i||^2 + ||\mathbf{v}_j||^2 + b_i^2 + b_j^2)$$

We can revise the update rule to be:

$$\mathbf{u}_i' = \mathbf{u}_i + \alpha \cdot (e_{ij} \cdot \mathbf{u}_i - \lambda \cdot \mathbf{v}_j)$$

$$\mathbf{v}_j' = \mathbf{v}_j + \alpha \cdot (e_{ij} \cdot \mathbf{v}_j - \lambda \cdot \mathbf{u}_i)$$

$$b_i' = b_i + \alpha \cdot (e_{ij} - \lambda \cdot b_i)$$

$$b_j' = b_j + \alpha \cdot (e_{ij} - \lambda \cdot b_j)$$

This is the model that we implemented for analyzing movie recommendations.

4 Empirical Results

4.1 MovieLens Data

The MovieLens[2] dataset includes a *ratings.csv* table, which looks like the following (first 5 rows):

```
   userId movieId rating  timestamp
1       1      31    2.5 1260759144
2       1    1029    3.0 1260759179
3       1    1061    3.0 1260759182
4       1    1129    2.0 1260759185
5       1    1172    4.0 1260759205
```

The full table has over 24 million ratings from 40K movies and 260K users. For this project, we used a smaller subset of the ratings table. In particular, we looked at 671 users and 3876 movies. The sparsity of the ratings matrix is about 5.7%. Note that the smaller dataset means that the algorithm has less information to learn from and hence the recommendations will be of lower quality, all else equal, simply due to lack of data. In practice we would use as much data as possible.

4.2 Evaluation

We ran the SGD learning algorithm on the above dataset for 5 different factor values, $K \in \{2, 5, 10, 20, 50\}$. We will evaluate the model in 3 ways:

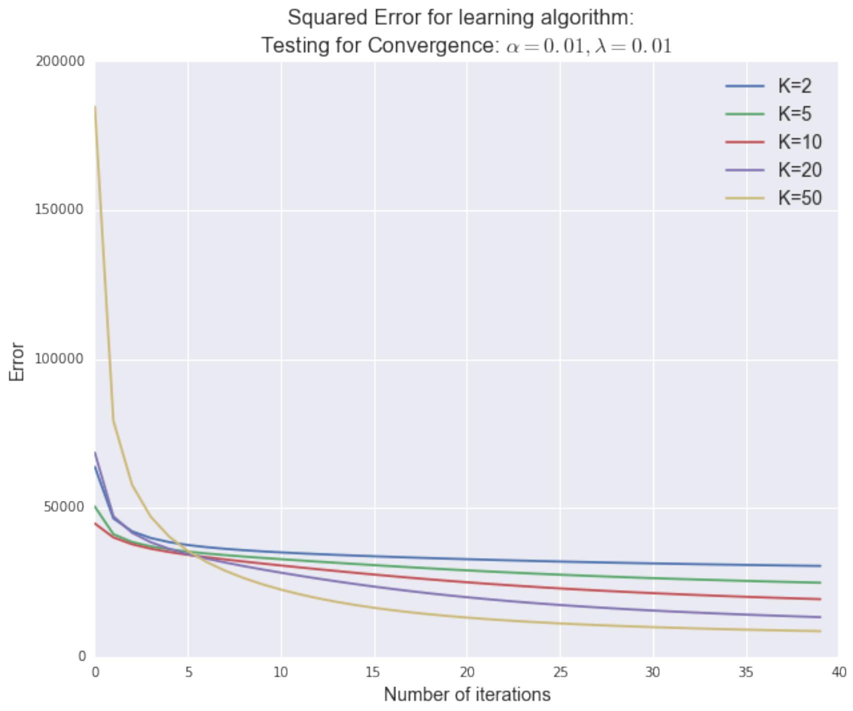
1. In Sample Error - How well does the model predict known ratings, from the training set?
2. Out of Sample Error - How well does the model predict unknown ratings, from the held out set?
3. Actual new movie recommendations - Does the model recommend new movies to users in a sensible way?

We use Root Mean Square Error for evaluating 1. and 2. above:

$$RMSE = \sqrt{\frac{1}{n}\sum(r_{ij} - \hat{r}_{ij})^2}$$

For 3. we rely on the author's subjective opinion. Finally, we set the learning parameter, α and regularization parameter λ each to 0.01.

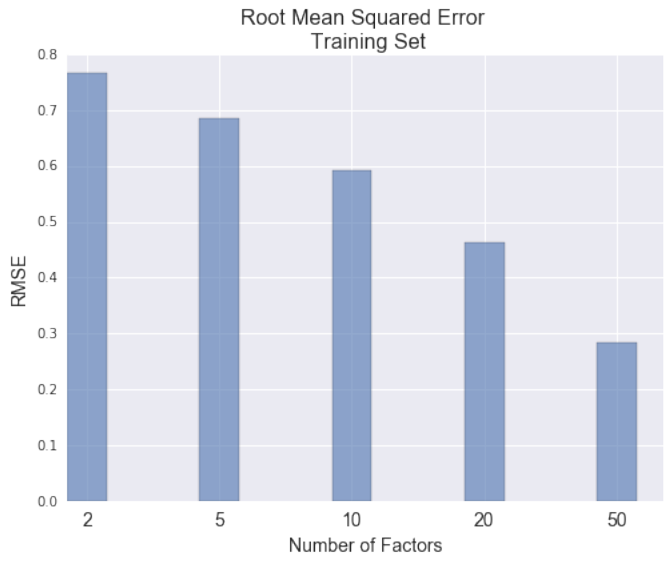
First we look at the convergence of the SGD algorithm for each factor value:



The squared error for the SGD algorithm starts to level off early on, indicating that we don't need that many iterations until convergence.

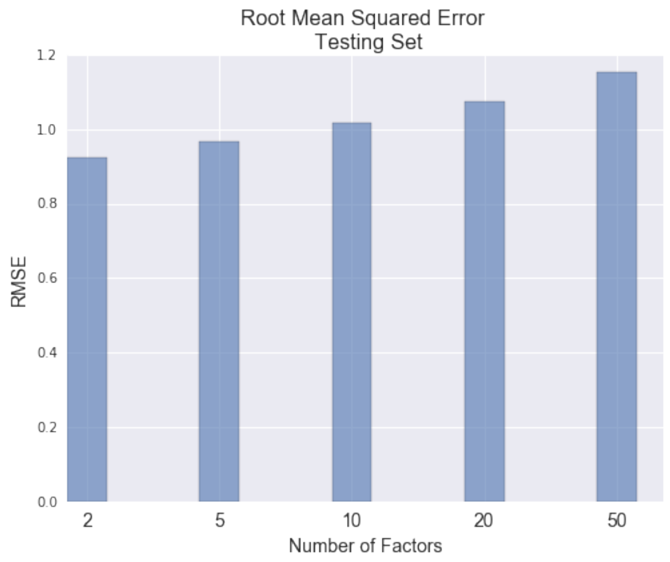
4.2.1 In Sample Error

The in sample, or training, error decreases as K increases. This makes sense as we expect a higher K to get closer and closer to the original matrix.



4.2.2 Out of Sample Error

However, the more instructive out of sample, or testing, error indicates that as K increases, the MSE gets worse. This indicates that perhaps the model is overfitting to the training data. This could be alleviated by a higher regularization penalty.



In practice we prefer a model with lower testing error than a model with lower training error. Therefore we proceed with a 2-factor model for making movie recommendations.

4.2.3 Movie Recommendations

We randomly selected users and evaluated the model's recommendations. For the purposes of this paper, we analyze a handful of (random) recommendations:

`Userid: 3`

- Highly rated movies: `Angel and the Badman, Dirty Dancing, RocketMan, The Outsiders, Diebinnen`
- Model recommended movies: `Moonlight and Valentino, Highlander: The Final Dimension, Emma`

This user appears to enjoy dramas and older movies. Of these three recommendations, 2 are dramas, and all of them were made in the mid 90's, though the `Highlander` seems a little out of place. However these are a small subset of movies that the user watched. There could well be other older, fantasy action movies in their highly rated list, making the `Highlander` a good recommendation.

`Userid: 20`

- Highly rated movies: `I'll Do Anything, Brothers in Trouble, An Affair to Remember, Dangerous Ground`
- Model recommended movies: `Calendar Girl, Freddy's Dead: The Final Nightmare, A Civil Action`

This user enjoys older romantic and dramatic comedies. The model recommends 2 dramas, 1 of which is a comedy. `Freddy's Dead` obviously doesn't fit the mold of the other movies, except for the fact that it's classified as a comedy (in addition to horror and thriller). Perhaps this user has an eclectic sense of comedy.

`Userid: 150`

- Highly rated movies: `Sudden Death, Up Close and Personal, Swimming with Sharks`
- Model recommended movies: `Leaving Las Vegas, Country Life, Nothing Personal`

This user appears to highly rate many types of movies - action, drama and comedy. This is likely to be the case for many people, i.e. people like all sorts of movies. This variety is what makes recommendations very difficult. Increasing the amount of data is one possible way to deal with these types of users because access to more data allows the model to learn more nuanced factors.

In general, our latent factor recommender system isn't perfect, but certainly can make some sensible recommendations. This could be due to a small dataset without enough training examples or due to a model that isn't picking up enough of the user-movie covariance in the data. This is something to investigate for future analysis.

5 Conclusion

This paper explored a latent factor model approach for building a recommender system based on collaborative filtering. We learned the matrix using stochastic gradient descent and extended the model by incorporating user and item biases, yielding an AMMI model. It appears that the number of latent factors doesn't improve RMSE on the testing set, so we used a model with $K = 2$. Using such a model, we explored individual movie recommendations, some of which made intuitive sense and some didn't.

In practice there are many different extensions to the model above. One popular extension is to include temporal affects, since user preferences and item popularity change over time. This is particularly interesting when analyzing data that is seasonal, e.g. preference for certain types of products may change with the weather, and is something we intend to explore in the future.

Another consideration is trying to discriminate between very low and very high ratings. For example, we could change the loss function from squared error, which penalizes errors in either direction equally, to a function which penalizes being incorrect in one direction more than another.

Clearly there is much demand in the space of recommendation engines and the literature is growing commensurate with that demand. As we add to the plethora of data out there, companies will be able to optimize their recommendations more and more precisely. We are excited to investigate future models and build more sophisticated and useful recommender systems.

References

[1] Yehuda Koren, Robert Ball, and Chris Volinsky *MATRIX FACTORIZATION TECHNIQUES FOR RECOMMENDER SYSTEMS*
Click here for paper

[2] GroupLens Research *MovieLens*
Click here for MovieLens