Recommender Systems For Movies

Chan-Ching Hsu, and Taewoon Kim[1]

1. Introduction
    1.1 Recommender Systems
2. Recommendation Platform
    2.1 Problem Formulation
3. Gradient-based Approaches
    3.1 Gradient Descent (GD)
    3.2 Alternating Least Squares (ALS)
4. Performance Evaluation
    4.1 Evaluation Metrics
        Measuring the prediction accuracy
        Measuring the recommendation quality
    4.2 Small Data Set
        4.2.1 Results from GD
        4.2.2 Results from ALS
    4.3 Larger Data Set
        4.3.1 Results from GD
        4.3.2 Results from ALS
        4.3.3 Comments on the Results
5. Parameter Analysis
    5.1 Tuning Iteration Limit
    5.2 Tuning Regularization Term
    5.3 Tuning Feature Number
6. Conclusion
    6.1 Future work
7. References
Appendix A. List of Codes

[1] Names are presented in an alphabetic order.

1. Introduction

Recommender systems (or recommendation systems) are a subclass of information filtering systems that seek to predict the 'rating' or 'preference' that a user would give to an item [1]. They have been used in various applications, such as movie, music, books, products and so on. Simply speaking, the recommender system can be applied to and be beneficial to any applications where sellers would like to trigger users' interest for purchasing items by recommending relevant items to users. The goal of the recommender system is, in other words, to find products that are most relevant to the users' interest given the information collected or provided by users. Although there are numerous ways to define relevance, one common way is to investigate preferences (for example, ratings to a variety items) that users revealed. For example, if a user shows a tendency of rating a bunch of viewed action movies high, then it is reasonable to assume that the user like action movies more that those in other genres. In this case, we would like to find out what other action movies that have been rated or viewed by the user may interest him/her. The recommender system is known to be good to both providers and consumers; the recommender system helps users who are looking for "similar" products to better filter out irrelevant information, and on the other hand, the provider can possibly gain more profit resulting from recommending the right things to the right customers. As the electronic commerce being more widely spreaded, there have been increasing attentions to recommender systems in the recent years [2][3].

In general, many of the proposed variants stem from two types of methodologies in building a recommender system: one is collaborative filtering and the other is content-based filtering -- the hybrid model combining these two approaches is also being studied and explored. In the next section, we discuss the aforementioned two basic approaches as an overview of recommender systems.

1.1 Recommender Systems

The content-based filtering approach works based on the characteristics/features of products and the user's previous purchase history or ratings given to some items. Based on the rationale that a user who likes a certain item might as well like other items sharing the similar characteristics/features, the recommender system says:

"If you like this item, you might also like the items below...
(because they have similar features as the one you preferred)."

The content-based filtering is using "local" information to make recommendations. By local information it means that recommendations are derived upon the user's purchase history or/and preferences.

The collaborative filtering approach, on the other hand, is to utilize the data (purchase history and the ratings information) of other customers. It examines decisions made by not just a specific user but all users for recommendation-making tasks. Given the information related to a particular item and user, the recommender system finds items that other users liked, and says.

"Customers who liked this item also liked the items below…"
or "Customers who share similar interests also liked the items below…"

This collaborative filtering operation is more complicated than the content-based one because it works with information in a larger degree. The advantage of the collaborative filtering approach is that the system does not need to "understand/comprehend" the products, which is in contrast to the content-based filtering approach. The content-based filtering system needs enough amount of the information of each products so that it can tell whether two given items are relevant or not, while the collaborative filtering approach investigates the feature similarity among users, so it requires less (or maybe little) information about products of interest.

In both approaches, beside the purchase history, the most common method to evaluate user preferences is by user ratings, because it provides a direct way to learn information about whether users' interests in items. More specifically, the rating information plays an important role in understanding users who share similar interests, which is often time the basis of building a recommender system. The rating data is one of the most obvious and straightforward information help study users' preferences; however user preference to items can be and have been acquired via various ways.

In this work, we implemented the collaborative filtering approach to estimate user ratings about movies that users have not rated yet in an available database. The estimate of ratings then is used for making movie recommendations to users. This article is organized as follows. The implemented platform of a recommender system is described in Section 2, followed by a discussion on the algorithms implemented to build the recommendation system. The performance of the developed recommender is evaluated in Section 4, where the system is evaluated on a small and large data sets. In Section 5, we further study the performance of the algorithms implemented from the parameter tuning perspective. And Section 6 concludes our work and shares a direction of future work.

2. Recommendation Platform

The aim of the desired movie recommender platform is to create a system that can take a given user rating data set about movies and generate a list of movie recommendations for a customer that have some rating records input to the interested dataset. We used MATLAB to implement our recommender system. The set of data to begin with is available from GroupLens Research [4]. The data set is called "MovieLens 100K Dataset" and is summarized as follows:

- 100,000 ratings (scaled from 1 to 5) obtained from 943 users on 1682 movies.
    - 5: the highest rating
    - 1: the lowest rating
- Each user rated at least 20 movies.

The main data files that are useful for our purpose include:

- `u.info` : description of the dataset (e.g., the number of users, movies, the the rated movies)
- `u.data` : 100K movie ratings labeled by (user id), (movie id), (rating), and (timestamp)
- `u.item` : full list of the movies including name and year

Two important matrices are constructed for further operations, named Y and R, both of which are with a dimension 1682-by-943:

- *Y(m,u)* is the rating entered by the user indexed by *u* (referred to as user-*u*) for the movie indexed by *m* (referred to as movie-*m*); a rating of 0 represents no rating information for the particular movie/user pair.
- *R(m,u)* is 1 of movie-*m* is rated by user-*u*; 0 otherwise.

Each value in Y is integer ranging from 1 to 5, while each value in R is binary, either 0 or 1. The Y matrix is sparse as expected -- each user graded roughly 20 movies.

Given Y and R matrices, we implemented the collaborative filtering approach to first, estimate the ratings of the movies that have not been rated by any users, and then we can make recommendations of movies to users based on the predicted ratings. In the same data set, the characteristics of each movies (i.e., genres) are also available; however, in this project since the collaborative filtering system does not rely on these

information, we leave that information for future work on a content-based filtering or enhanced hybrid version of these two approaches.

2.1 Problem Formulation

We first denote the following variables:

- *Nm* : number of movies available
- *Nu* : number of users registered
- *Nf* : number of features (design parameter)
- *X* : characteristic of movie, dimension is *Nm-by-Nf,* with randomized values initially
- *Theta* : preference of users, dimension is *Nu-by-Nf,* with randomized values initially

From the given movie-user rating information, the movie features, i.e., X, and user preferences, i.e., Theta, will be learned. Let X(m,:) be the feature vector of movie-*m*, and Theta(u,:) be the preference vector of user-*u*, both of which will be learned by the recommender system. If movie-*m* has not been rated by user-*u*, the rating of the movie is estimated by X(m,:) * Theta(u,:)'; the corresponding entries of the resulting product are the predicted ratings for the movie-user pairs.

Based on the existing rating records, we want to find both X and Theta matrices, which are movie characteristic and user preference vector, respectively, so that when compare the predicted ratings (i.e., X * Theta') and user actual ratings, the overall difference over all elements(i.e., Y(m, u) - (X(m, u) * Theta(u, m)')) is as small as possible. This results in the following minimization problem:

$$J(X,\Theta) = \sum_{(i,j):R(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + reg(\Theta) + reg(X),$$

where

- J(.) is the cost function, which evaluates the difference between the given ratings from users and the estimated ratings derived from X and Theta.
- The summation is performed only over rated items; in other words, only for the entries in R have the value equal to 1, we evaluate the difference.
- It is a general linear regression model with regularization terms, which are defined in the following, where lambda is the weight of the regularization terms:
    - $reg(\Theta) := \frac{\lambda}{2} \sum_{j=1}^{Nu} \sum_{k=1}^{Nf} (\theta^{(j)}{}_k)^2$

- $reg(X) := \frac{\lambda}{2}\sum_{i=1}^{Nm}\sum_{k=1}^{Nf}(x^{(i)}{}_k)^2$

As the function is smooth, the optimal solution can be obtained to solve the formulated problem. We can derived the gradients with respect to X and Theta as follows:

- $\frac{\partial J}{\partial x^{(i)}{}_k} = \sum_{j:R(i,j)=1}((\theta^{(j)})^T x^{(i)} - y^{(i,j)})\theta^{(j)}{}_k + \lambda x^{(i)}{}_k$
- $\frac{\partial J}{\partial \theta^{(j)}{}_k} = \sum_{i:R(i,j)=1}((\theta^{(j)})^T x^{(i)} - y^{(i,j)})x^{(i)}{}_k + \lambda \theta^{(j)}{}_k$

The overall procedure of recommender systems can be described in the following steps.

1. The decision variables, both X and Theta are randomized initially.
2. Solve the formulated problem to train a classifier, obtaining the two decision variables, X and Theta.
3. With X and Theta, the system estimates movie ratings for users and makes recommendations.

3. Gradient-based Approaches

In this section, we present two gradient-based methods used to solve X and Theta so that the cost function is minimized. In general, a gradient-based algorithm can be described in the algorithm below:

```
GIVEN a starting point x(0) ∈ dom f
REPEAT
   1. Set the direction: Δx
   2. Choose the step size: t
   3. Update: x(r+1) := x(r)+tΔx
UNTIL stopping criterion is satisfied.
```

Based on the same philosophy, we implemented the following two methods.

3.1 Gradient Descent (GD)

We find the optimal X and Theta by running a gradient based algorithm that was created by Prof. Carl Edward Rasmussen in Cambridge university. The algorithm minimizes a continuous differentiable multivariate function. In short, the Polack-Ribiere flavour of conjugate gradients is used to compute search directions, and a line search using quadratic and cubic polynomial approximations and the Wolfe-Powell stopping criteria is used together with the slope ratio method for guessing initial step sizes.

3.2 Alternating Least Squares (ALS)

Without the regularization terms, the collaborative filtering problem can be described as :

$$min_{\{X,\Theta\}} \|X\Theta^T - Y\|^2.$$

Due to the fact that the two decision variables are coupled, the given least square problem is not convex. However, the same problem become convex if one of the decision variables is fixed; that is, given X fixed, the problem is convex because the only decision variable is Theta, and vice versa. The general description of the ALS algorithm is as follows [5][6]:

Algorithm:

```
1. Initialize X and Theta.
2. Fix X, and optimize Theta by minimizing the cost function.
3. Fix Theta, and optimize X by minimizing the cost function.
4. Repeat Steps 2 and 3 until the cost function converges.
```

4. Performance Evaluation

In this section, we describe first how we measure the performance of the developed recommendation system, and then evaluate the performance using different gradient approaches discussed in the previous section.

4.1 Evaluation Metrics

Measuring the prediction accuracy

In order to measure the accuracy of the recommender system, the metric of Mean Squared Error (MSE) is adopted, which is defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\widehat{Y}_i - Y_i)^2,$$

where n is the number of elements for which we make predictions, $\widehat{Y}_i$ represents predictions made by the system evaluated, and Y_i is the observed values (or the *true* values). To measure the MSE, we divided the given user ratings into training and test data sets; the training set is used for learning X and Theta, whereas the rated items in the test set are the observed values with which we compare the predicting ratings and calculate the MSE. To estimate the error, two commonly used techniques are applied, hold-one-out and k-fold cross-validation. Generally, the evaluation procedure is as follows:

1. Take out rating records for each user.
2. Learn the parameters: X, Θ
3. Predict the ratings of those movies whose records have been taken out
4. Measure the metrics

Please note although some work use Root Mean Squared Error (RMSE) to evaluate recommender systems, it is straightforward to obtain RMSE from MSE.

Measuring the recommendation quality

For contexts where ranked list of items are produced and item rankings can be extracted from the user's rating data, the algorithm's capacity to produce rankings that match user's can be measures.

Two ways of measuring this are with Pearson correlation or Spearman rank correlation coefficients. As noted by Herlocker et al. [7], however, the Spearman rank correlation metric suffers when the recommender makes distinctions between items that the user ranks at the same level. These metrics also penalize inaccuracy in the tail of the recommendation list, which the user will likely never see, to the same degree as inaccuracy in the top-predicted items.

Another metric, that is also used in this work, for evaluating ranking accuracy is the *half-life utility* metric [8]. It measures the expected utility of a ranked recommendation list, based on the assumption that users are more likely to look at items higher in the list; this assumption is reasonable for many real systems such as e-commerce sites. It requires two parameters: a half-life α, such that the probability of a user viewing a recommendation with rank α is 0.5, and a default rating d. d should be selected to indicate neutrality, providing neither benefit for the user; it can be the user's mean rating, the system's overall mean rating, or some fixed neutral/ambivalent point in the rating scale.

The half-life expected utility R_u of the recommendation list for a user u is defined in the following equation. k_i is the 1-based rank at which item i appears. Throughout our experiments, $d = 3$ and $\alpha = 5$.

$$R_u = \sum_i (max(y_{u,i} - d,\ 0)/2^{(k_i - 1)/(\alpha - 1)})$$

In order to measure the performance of a recommender across users, this metric is normalized by the maximum achievable utility R_u^{max} for each user, resulting in a value in [0, 1] representing the fraction of potential utility achieved. R_u^{max} is the utility achieved by listing the best items for the user in order. The normalization is needed to compensate for different users having different potential utilities; one way this can happen is as a result of some users having more purchases to predict than others. The overall score R is given by the equation:

$$R = (\sum_u R_u)/(\sum_u R_u^{max}).$$

4.2 Small Data Set

4.2.1 Results from GD

We first solved the Recommender System problem by using the Gradient Descent algorithm.

<Parameter configuration>

PARAM	VALUE	MEANING
MAX_ITER_FMINCG	20	The maximum number of iteration
num_features	50	The number of features
lambda	5	The weight to the regularization term

<Result: MSE>

```
0.229436
```

We used hold-one-out technique to estimate the accuracy by hiding 944 movie ratings from the original data set (one from each user). We then compared the estimated results of those hided ratings to the actual ratings given by the users. As we can see from the MSE result (0.229436), the estimated ratings were very accurate.

<Estimated ratings vs. actual ratings>

As an example, we illustrate the accuracy of the estimated ratings by estimating the movie ratings for those movies that are observed from a certain user. We added a user into the original data set along with some ratings as below:

```
The user ratings:
Rated 4 : Toy Story (1995)
Rated 2 : Twelve Monkeys (1995)
Rated 2 : Usual Suspects, The (1995)
Rated 4 : Outbreak (1995)
Rated 2 : Shawshank Redemption, The (1994)
Rated 1 : While You Were Sleeping (1995)
Rated 3 : Forrest Gump (1994)
Rated 1 : Silence of the Lambs, The (1991)
Rated 5 : Godfather, The (1972)
Rated 4 : Alien (1979)
Rated 5 : Die Hard 2 (1990)
Rated 5 : Sphere (1998)
Rated 4 : Time Tracers (1995)
Rated 5 : New York Cop (1996)
Rated 4 : Surviving the Game (1994)
Rated 4 : Men With Guns (1997)
```

After prediction procedure, we "estimated" those ratings for which the actual ratings are observable in order to check the accuracy of the movie ratings. Please note that the estimations are not going to be actually used in real systems because those are already rated by the user. Below is the comparison between the actual ratings provided by the same user and the ones estimated by the system:

```
Original ratings provided vs. estimated:
4.00 vs 3.97 ... Toy Story (1995)
2.00 vs 2.01 ... Twelve Monkeys (1995)
2.00 vs 2.21 ... Usual Suspects, The (1995)
4.00 vs 3.73 ... Outbreak (1995)
2.00 vs 1.57 ... Shawshank Redemption, The (1994)
1.00 vs 1.37 ... While You Were Sleeping (1995)
```

```
3.00 vs 3.19 ... Forrest Gump (1994)
1.00 vs 1.50 ... Silence of the Lambs, The (1991)
5.00 vs 5.01 ... Godfather, The (1972)
4.00 vs 4.06 ... Alien (1979)
5.00 vs 4.79 ... Die Hard 2 (1990)
5.00 vs 4.89 ... Sphere (1998)
4.00 vs 3.72 ... Time Tracers (1995)
5.00 vs 4.72 ... New York Cop (1996)
4.00 vs 3.78 ... Surviving the Game (1994)
4.00 vs 3.89 ... Men With Guns (1997)
Mean squared error: 0.062595
```

The estimation on those given ratings is much accurate (MSE is only 0.062595) because the objective of the optimization problem we solve is to minimize the difference between the estimation and the provided ratings.

<Recommendations>

Given the estimated ratings for the movies that are not rated, the system selected the top 10 movies based on the estimated ratings into the recommendation list for the customer:

```
Top 10 recommendations for you:
1. Everyone Says I Love You (1996)
2. Godfather: Part II, The (1974)
3. GoodFellas (1990)
4. Four Weddings and a Funeral (1994)
5. Pulp Fiction (1994)
6. Peacemaker, The (1997)
7. Taxi Driver (1976)
8. Heathers (1989)
9. Postino, Il (1994)
10. Streetcar Named Desire, A (1951)
```

4.2.2 Results from ALS

Similarly, we present the performance of ALS.

<Parameter configuration>

PARAM	VALUE	MEANING
MAX_ITER_ALS_OUTER	10	The maximum number of the outer iterations (i.e., the iteration limit for the overall procedure)

MAX_ITER_ALS_INNER	1	The maximum number of the inner iterations (i.e., the iteration limit for each of the partial-gradient algorithm for either X or Theta)
num_features	50	The number of features
lambda	n/a	ALS does not have the regularization term

Note that since ALS does not have any regularization terms, the lambda does not have any effect on the performance of ALS. Also we set the MAX_ITER_ALS_INNER to 1 so that both the gradient descent algorithm and ALS can be fairly compared under the same iteration limit. The maximum number of iterations that the gradient descent algorithm could take was 20; the same number of iterations will be applied to ALS as [MAX_ITER_ASL_OUTER] X [MAX_ITER_ALS_INNER] X [Number of decision variables to be considered] is 20 as well.

<Result: MSE>

```
0.252474
```

<Estimated ratings vs. actual ratings>

The same scenario in the previous section was also applied; the same user and same ratings were added. The difference between the given movie ratings and the estimated movies ratings are:

```
Original ratings provided vs. estimated:
4.00 vs 3.70 ... Toy Story (1995)
2.00 vs 3.43 ... Twelve Monkeys (1995)
2.00 vs 3.30 ... Usual Suspects, The (1995)
4.00 vs 3.57 ... Outbreak (1995)
2.00 vs 3.02 ... Shawshank Redemption, The (1994)
1.00 vs 1.47 ... While You Were Sleeping (1995)
3.00 vs 2.96 ... Forrest Gump (1994)
1.00 vs 2.28 ... Silence of the Lambs, The (1991)
5.00 vs 4.67 ... Godfather, The (1972)
4.00 vs 4.36 ... Alien (1979)
5.00 vs 4.54 ... Die Hard 2 (1990)
5.00 vs 4.91 ... Sphere (1998)
4.00 vs 3.80 ... Time Tracers (1995)
5.00 vs 4.82 ... New York Cop (1996)
4.00 vs 4.01 ... Surviving the Game (1994)
4.00 vs 3.87 ... Men With Guns (1997)
Mean squared error: 0.466725
```

<Recommendations>

Given the estimated ratings of the movies that are not rated by the user, the top 10 movies was selected based on the estimated ratings:

```
Top 10 recommendations for you:
1. Umbrellas of Cherbourg, The (Parapluies de Cherbourg, Les) (1964)
2. Faces (1968)
3. Great Dictator, The (1940)
4. Microcosmos: Le peuple de l'herbe (1996)
5. Anne Frank Remembered (1995)
6. Amateur (1994)
7. Clockwork Orange, A (1971)
8. Affair to Remember, An (1957)
9. Jeffrey (1995)
10. Maltese Falcon, The (1941)
```

4.3 Larger Data Set

We further evaluate the designed recommender system with a larger data set which is also made public by the GroupLense Research Project. The data set is "MovieLense 1M Dataset" which has 1,000,209 ratings from 6,040 users on 3,900 movies (released 2/2003). We employed the same configurations as on the small data set to evaluate the performance of the system. It is noteworthy that there are other data sets available with much bigger sizes, however, due to the memory limitation of the testbed, we could not load those data set for further study. In addition, because of the increased problem size, the execution time of finding the optimal values considerably increases.

We expect the performance of the recommender system will be affected with a larger data set because the matrices R and Y are even more sparse. A comparison between the two datasets are given as follows.

<u>The 100K dataset:</u>

- # users: 943
- # movies: 1682
- The number of data points = # users * # movies = 1,586,126
- The mean number of ratings provided by each user: 20
- The mean number of ratings provided = # users * 20 = 18,860
- The percentage of mean ratings given = 18,860 / 1,586,126 = **1.189%**
- The actual # ratings given: 100K

- The percentage of ratings given = 100K / 1,586,126 = **6.3%**

The 1M dataset:

- # users: 6040
- # movies: 3952
- The number of data points = # users * # movies = 23,870,080
- The mean number of ratings provided by each user: 20
- The mean number of ratings provided = # users * 20 = 120,800
- The percentage of mean ratings given = 120,800 / 23,870,080 = **0.506%**
- The actual # ratings given: 1M
- The percentage of ratings given = 1M / 23,870,080 = **4.2%**

The lack of information is expected to degrade the accuracy of estimation, but not much hopefully.

4.3.1 Results from GD

We start by introducing the performance of the gradient descent method over the larger data set.

<Result: MSE>

```
0.402909
```

<Estimated ratings vs. actual ratings>

The same scenario from the previous section again was applied with a new user and the same movie ratings. The difference between the given movie ratings and the estimated movies ratings are:

```
Original ratings provided vs. estimated:
4.00 vs 4.09 ... [movie index: 0001]
2.00 vs 2.47 ... [movie index: 0007]
2.00 vs 2.55 ... [movie index: 0012]
4.00 vs 3.70 ... [movie index: 0054]
2.00 vs 2.05 ... [movie index: 0064]
1.00 vs 1.35 ... [movie index: 0066]
3.00 vs 2.88 ... [movie index: 0069]
1.00 vs 1.19 ... [movie index: 0098]
5.00 vs 4.54 ... [movie index: 0127]
4.00 vs 3.79 ... [movie index: 0183]
5.00 vs 4.53 ... [movie index: 0226]
5.00 vs 3.98 ... [movie index: 0355]
```

```
4.00 vs 3.86 ... [movie index: 0897]
5.00 vs 4.40 ... [movie index: 1304]
4.00 vs 3.97 ... [movie index: 1314]
4.00 vs 3.58 ... [movie index: 1646]
Mean squared error: 0.178117
```

4.3.2 Results from ALS

Next, we show the performance of ALS over the larger data set.

<Result: MSE>

```
0.594400
```

<Estimated ratings vs. actual ratings>

The difference between the given movie ratings and the estimated movies ratings are:

```
Original ratings provided vs. estimated:
4.00 vs 4.19 ... [movie index: 0001]
2.00 vs 3.12 ... [movie index: 0007]
2.00 vs 1.78 ... [movie index: 0012]
4.00 vs 3.55 ... [movie index: 0054]
2.00 vs 1.59 ... [movie index: 0064]
1.00 vs 1.17 ... [movie index: 0066]
3.00 vs 3.46 ... [movie index: 0069]
1.00 vs 0.66 ... [movie index: 0098]
5.00 vs 4.72 ... [movie index: 0127]
4.00 vs 4.09 ... [movie index: 0183]
5.00 vs 4.48 ... [movie index: 0226]
5.00 vs 3.32 ... [movie index: 0355]
4.00 vs 4.28 ... [movie index: 0897]
5.00 vs 4.43 ... [movie index: 1304]
4.00 vs 3.56 ... [movie index: 1314]
4.00 vs 3.05 ... [movie index: 1646]
Mean squared error: 0.421308
```

4.3.3 Comments on the Results

The decrease in the estimation accuracy is due to the more sparse matrices to work with. More precisely, the accuracy of the prediction on the hidden ratings for the larger data set is almost twice as much in percentage for the small data set. The gradient descent method outperformed ALS with the same configurations applied on the smaller data set. Also, the execution time increased a lot when the system was dealing with the larger data set. However, the MSE of estimation is less than 0.6 in both gradient and

ALS, showing that the developed system is able to estimate the movie ratings with a reasonable accuracy even when the amount of useful rating information is relatively smaller.

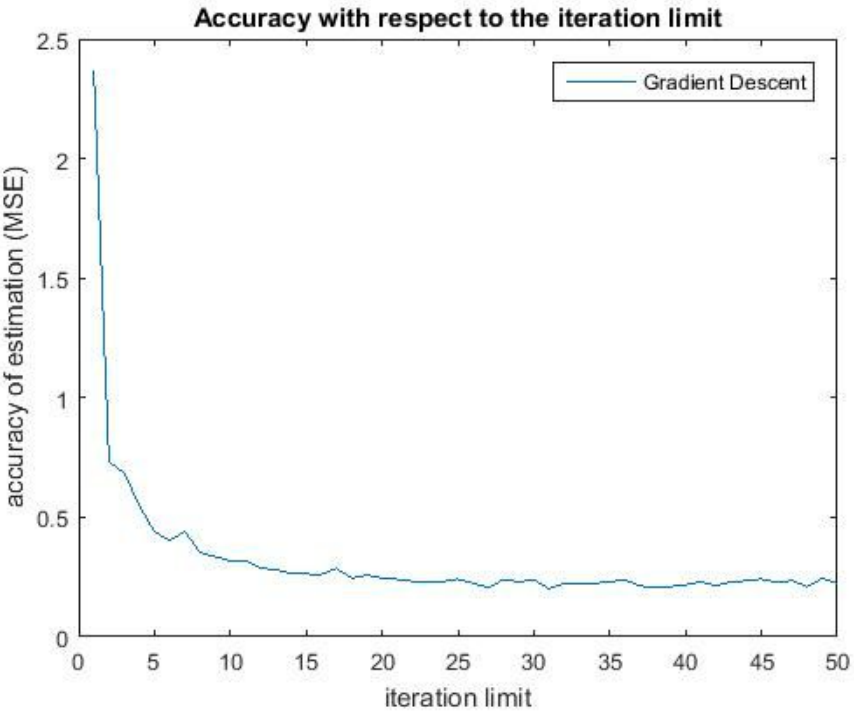
5. Parameter Analysis

In the algorithms, there are a few parameters that can be tuned for performance; therefore we are interested in knowing how does the accuracy/performance of the implemented algorithms change under different configurations. These parameters include the maximum number of iterations, the weight to the regularization term, and the number of features, whose results are presented in this section.

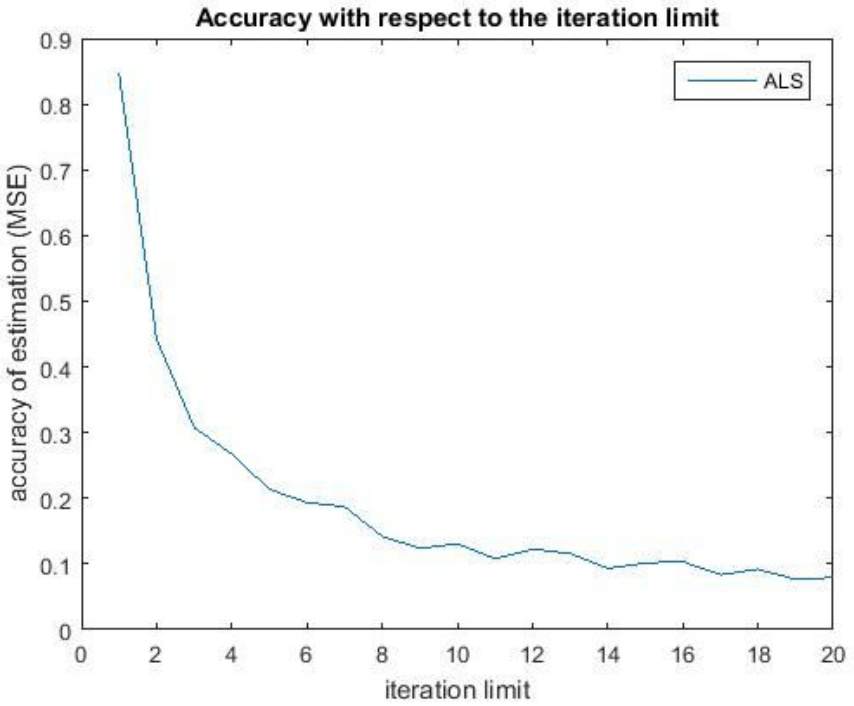
5.1 Tuning Iteration Limit

First, we study how the accuracy of estimation of both the gradient descent and ALS varies as changing the maximum iteration count. The other parameter configurations for both algorithms are the same as in the previous section except the iteration limit. The iteration limit increases from 1 to 50 in the case of the gradient descent algorithm, and 1 to 20 in the case of ALS. Note that we set the inner iteration count of ALS to 2, meaning that after fixing one variable, either X or Theta, we run two iterations for a variable. The maximum number of iterations that ALS can go through can be represented as `i X 2 X 2`, where `i` is the iteration limit that will be tuned, the first number of 2 corresponds to the number of decision variables, X and Theta, and the last number of 2 indicates the iteration limit for each variable.

<MSE of GD>


Accuracy with respect to the iteration limit
Gradient Descent
accuracy of estimation (MSE)
iteration limit
0
0.5
1
1.5
2
2.5
0
5
10
15
20
25
30
35
40
45
50

<MSE of ALS>


Accuracy with respect to the iteration limit
ALS
accuracy of estimation (MSE)
iteration limit
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
2
4
6
8
10
12
14
16
18
20

For both algorithms, the accuracy of the estimation gets better as the iteration limit increases, and it becomes more stable as the slope of the curves flatten out.. Therefore, one would increase the iteration limit in order to increase the system performance. Since the optimal decision variables do not need to be calculated in real time, the system operator can set a quite large number for the iteration limit when he/she is looking for the optimal solution offline.

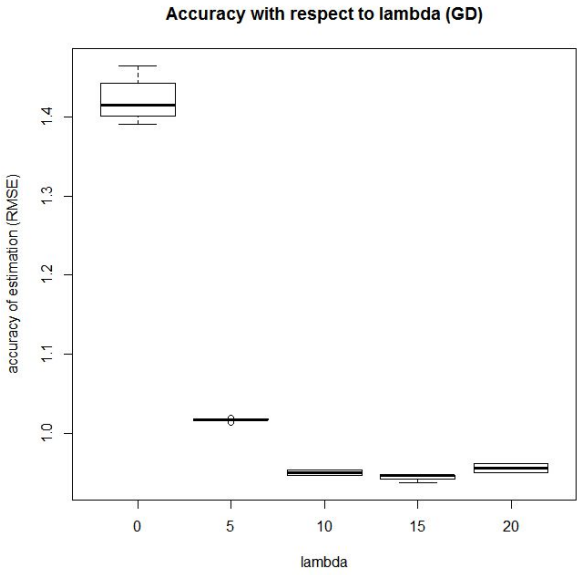
5.2 Tuning Regularization Term

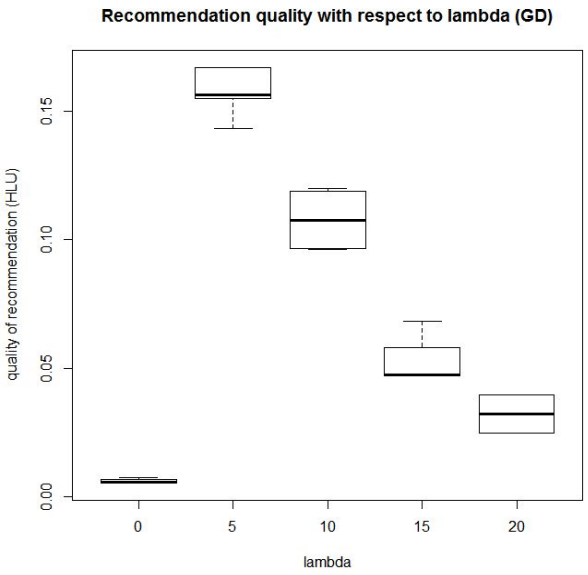
Following the result obtained in the experiment of tuning the parameter of the iteration limit, we selected the iteration limit of 20 for the GD algorithm and 20 for the ALS algorithm. A limit on iteration number at 20 seems reasonable because from the figure that shows the behavior of the GD algorithm in the previous section, the variation of accuracy appears to be small after iteration 20; on the other hand, the ALS algorithm reaches the lowest point at iteration 20.

Here we adopted cross-validation techniques with different number of folds to evaluate the performance for different lambda settings on the small data set. We compare in box plots the performance of the algorithms in terms of the accuracy and recommendation quality as mentioned in Section 4.1. Please note that the higher the half-life utility value, the better the performance, while the lower the accuracy of estimation, the better the performance.

<Results of GD>

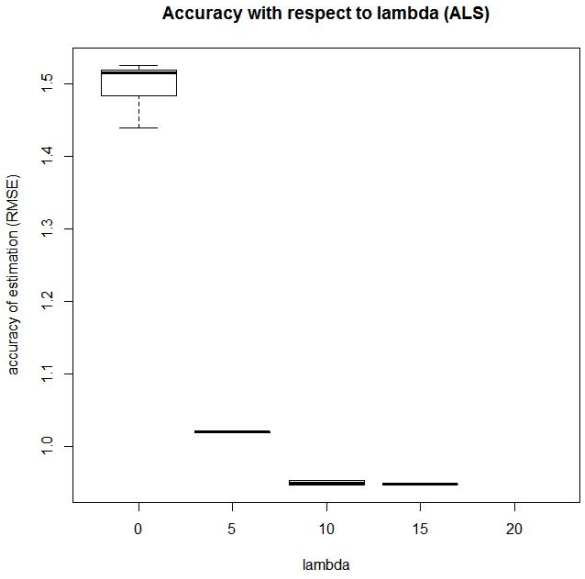
We first evaluate the performance of the GD algorithm and show the accuracy on the left panel and recommendation quality on the right panel in the figure below. Setting lambda to a value somewhere between 5 and 15 is an effective tuning strategy to the small data set to get a balance between accuracy and recommendation quality.

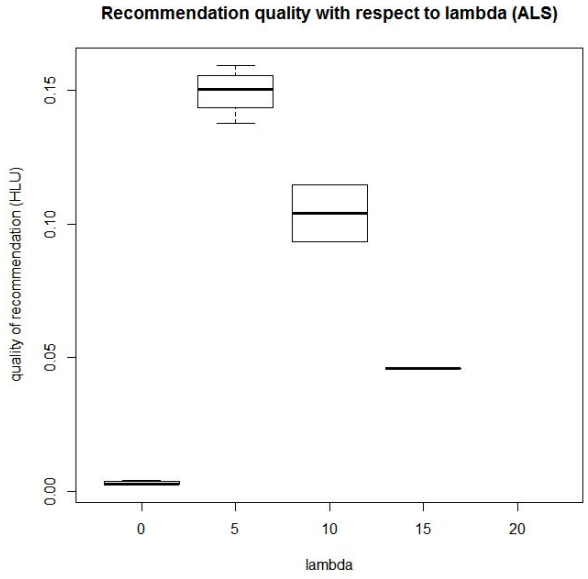




<Results of ALS>

The ALS algorithm behaves in a very similar way in terms of both metrics, suggesting the consistency and confirming the tuning parameter of lambda.





The value of lambda is expected to affect the optimal solution and hence the performance of algorithms. A higher lambda value penalizes more the solution vector, limiting the variable vector to be small, while a small lambda value puts less restriction on the utility function. Given the results of tuning iteration limits

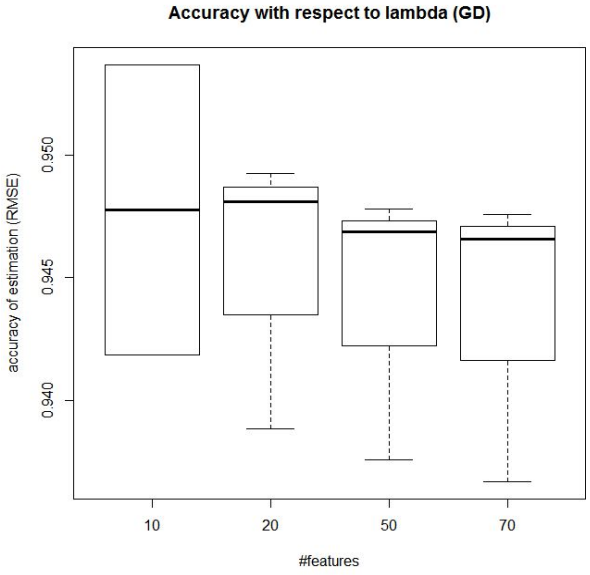
and regularization terms, we can select the parameters values which gave us the best performance for real use.

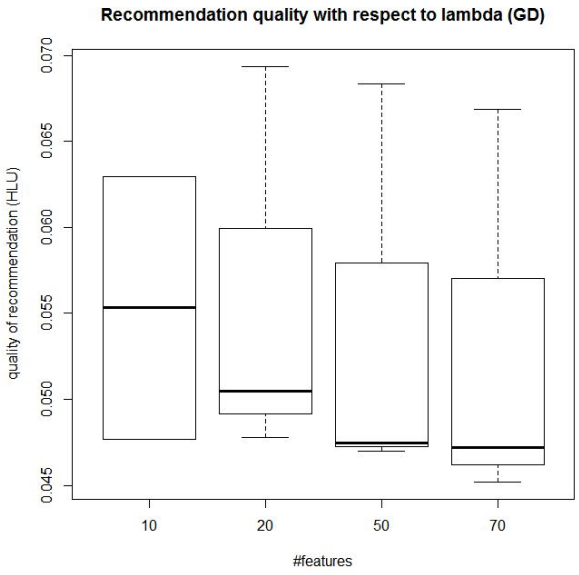
5.3 Tuning Feature Number

Here, we again selected 20 iteration limit and 15 as the lambda value as the RMSE was the lowest among all the tested values.

<Results of GD>

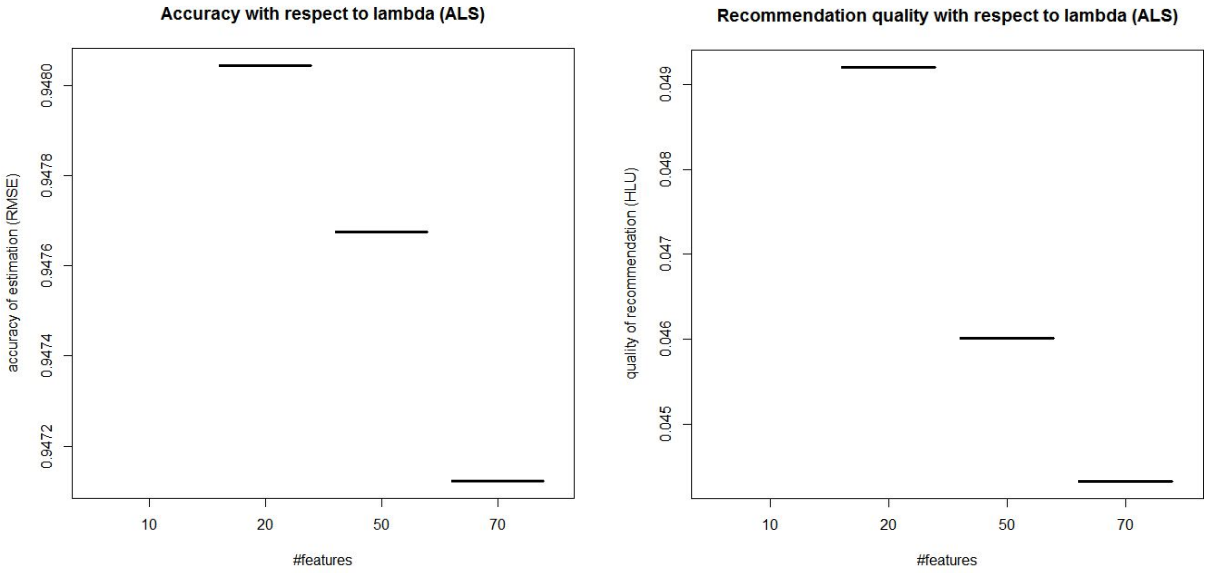
From the figures below, which show the accuracy and recommendation quality of the GD algorithm, it suggest that the number of features seems to have slight impact on performance; the variation is fairly low among different feature number settings.





<Results of ALS>

Similarly, the observed variations among different feature numbers are subtle for the ALS algorithm, whose results are presented in the figures below.



Based on the experiments carried out for tuning the parameter of feature numbers, we observe no particular "best" value for this parameter; however, the number should not be unreasonable high as we can see that a larger feature number does lead to a lower performance, although the difference is not profound. Therefore, for the small data set, a feature number no larger than 20 can be suggested.

6. Conclusion

We implemented a Recommender system using a collaborative filtering approach. The problem is formulated as a least square regression model with regularization terms, and to solve the problem, we implemented two algorithms, the gradient descent algorithm and the alternating least square algorithm. The movie rating datasets used in this work are made available by the GroupLens Research Project group, consisting of different numbers of rating records. With the linear regression model with regularization terms, our system can predict movie ratings of users which they did not rate. The performance of the Recommender system is shown. The results show that the system can estimate the movie ratings with a satisfactory accuracy on the datasets of interest.

6.1 Future work

In the given data set from the GroupLens Research project, a lot more information are available, such as:

- The timestamp when the rating has been given,
- The genre of each movie
- The demographic information about the users (e.g., age, gender, occupation and zip code),

With those information, we can derive additional information or increase the accuracy of the rate estimation. One possible solution is to take advantage of proximity of information in the given data set. For example, medical doctors are likely to view more medical movie, giving those more ratings to those types of movies. Or people at a similar age, for example, teenagers, are likely to be fond of teenage movies. Under this assumption, we probably can derive a more accurate and sophisticated system from the other sections of data. This direction is left as one of the future work.

The implemented algorithms suffer when the data set is in a very large size; the computation time increases exponentially as we experienced. To this regard, one approach is to decompose the problem into master problem and subproblems and solve the problems iteratively. The most straightforward benefit of this method is that we can reduce the complexity of the problem, potentially improving the computation time. For Recommender Systems, the complexity of the problem itself is not that difficult but the size of the data to deal with can be very huge, is the numbers of movies and users are massive, for example, Netflix. Thus, if we can apply a technique that decomposes the Recommender System problem into smaller, low-complex subproblems that can be efficiently solved in a distributed manner and maybe parallel, without degradation of the performance.

7. References

1. Wikipedia, Recommender system. https://en.wikipedia.org/wiki/Recommender_system.
2. J. Beel et al., Research paper recommendation system evaluation: a quantitative literature survey, *Proceedings of the workshop on reproducibility and replication in recommender system evaluation (RepSys) at the ACM recommender system conference (RecSys)*, 2013.

3. G. Adomavicius, Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions, *IEEE transactions on knowledge and data engineering*, 2005.
4. GroupLens, http://grouplens.org/datasets/movielens/
5. IE 587 Big Data Optimization, Lecture slides, Iowa State University.
6. Y. Zhou et. al., Large-scale parallel collaborative filtering for the Netflix prize, In *proceedings of the 4th international conference on algorithmic aspects in information and management*, AAIM 2008.
7. J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Transactions on Information Systems*, vol. 22, no. 1, pp. 5–53, 2004.
8. J. S. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *UAI 1998*, pp. 43–52, AAAI, 1998.

Appendix A. List of Codes

Below is the main list of the files for this work. Please refer to the project on GitHub for the up-to-date and complete list of files that we have, https://github.com/overegoz/ie587spr16.

1. **`cost_grad.m`** : calculates and returns both the cost function and the gradients (one for *X* and the other for *Theta*)
2. **`cost_gradX.m`** : calculates and returns both the cost function and the gradients for *X* only (used for ALS)
3. **`cost_gradTheta.m`** : calculates and returns both the cost function and the gradients for *Theta* (used for ALS)

4. **`data_build.m`** : read the `raw_data.txt` and build the *R* and *Y* matrices
5. **`fmincg.m`** : optimizer (minimizer) utility
6. **`read_movie_titles.m`** : load the list of the movie titles from `movie_ids.txt`
7. **`MAIN.m`** : main file which includes all the steps required to build the Recommender System for movie, steps included are: 1) loading the data set, 2) learning the parameters, 3) make some suggestions to a user, and so on.
8. **`movie_ids.txt`** : list of the movie titles
9. **`Movies.mat`** : R and Y data matrices generated by `data_build.m`
10. **`meanNormalization.m`** : normalize the ratings (if a user rated non of the movies, the estimated ratings of the movies for the user will become the mean rating of each movie)
11. **`raw_data.txt`** : movie rating data set from GroupLens Research Project