CSE 255 Assignment 1 : Movie Rating Prediction using the MovieLens dataset

Yashodhan Karandikar
ykarandi@ucsd.edu

1. INTRODUCTION

The goal of this project is to predict the rating given a user and a movie, using 3 different methods - linear regression using user and movie features, collaborative filtering and latent factor model [22, 23] on the MovieLens 1M data set [6]. We describe results using each of the 3 models and compare them with those obtained using the Apache Mahout machine learning library [1]. We also briefly discuss some results on restaurant ratings data from the Google Local data set [4].

2. THE MOVIELENS DATA SET

In this section, we introduce the MovieLens 1M data set [6]. This is a data set of about 1 million ratings from 6040 users on 3900 movies, thus containing around 165 ratings per user and 256 ratings per movie on an average. Ratings are integers on a 5-star scale. Each user and each movie is identified by a unique id. The data set includes information about the age (7 age groups), gender, occupation (21 types) and zip code for each user, as well as the title and genre (18 types) for each movie. Each user has at least 20 ratings.

We normalize the ratings so that each rating is in the closed interval $[0, 1]$. We perform an exploratory analysis of the data set by plotting the average rating against user and item features. This analysis is shown in figures 1, 2, 3, 4.

Figure 1 shows the average rating for each of the age groups. We observe a slightly increasing trend in the average rating as the age group of the user increases, suggesting that the age group of the user might be a useful feature in predicting movie ratings.

Figure 2 shows the average rating for male users and female users. We observe a very small difference between the average ratings for male and female users, suggesting that gender of the user is not likely to be a useful feature in prediction of movie ratings.

Figure 3 shows the average rating for users in various occupations. We observe that there is a reasonable amount of variation in the average ratings for various occupations, with the minimum and maximum average ratings being 0.681257 and 0.757114 respectively. This indicates that the occupation of the user might be a useful feature in prediction of ratings.

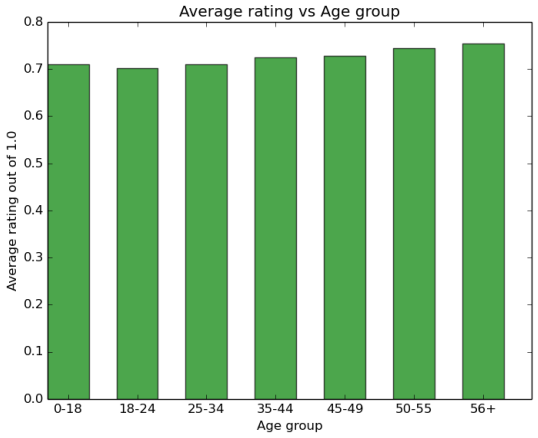

Figure 1: Average rating for different age groups

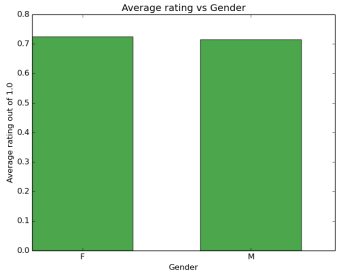

Figure 2: Average rating for male and female users

Figure 4 shows the average rating for various movie genres. We observe that there is a reasonable amount of variation in the average ratings for various genres, with the minimum and maximum average ratings being 0.644848 and 0.815781 respectively. This indicates that the genre of the movie might be a useful feature in prediction of ratings.

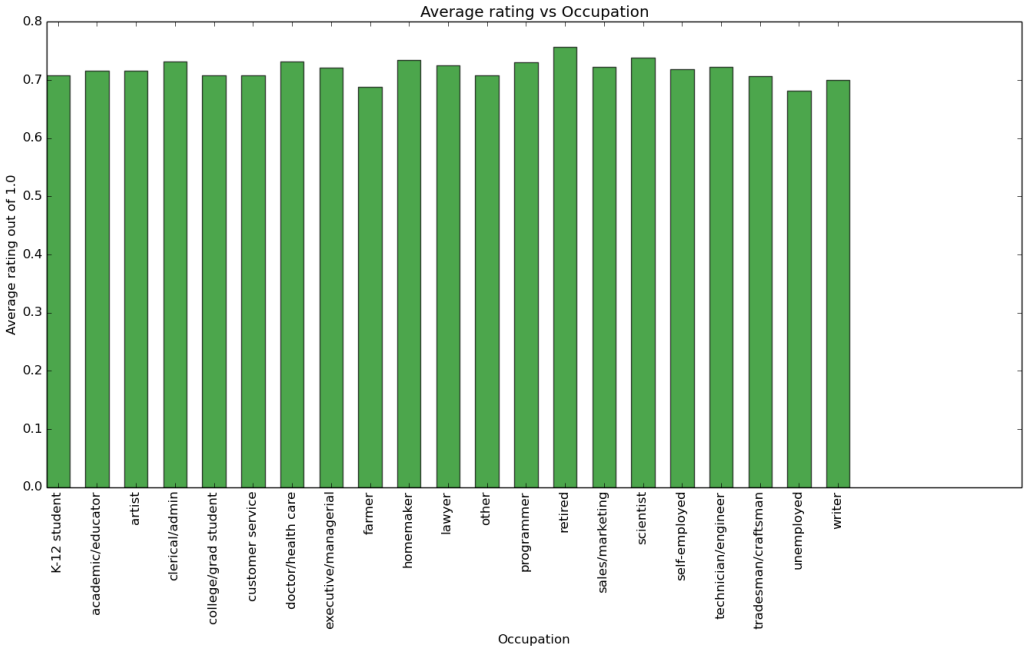

Figure 3: Average rating for various occupations

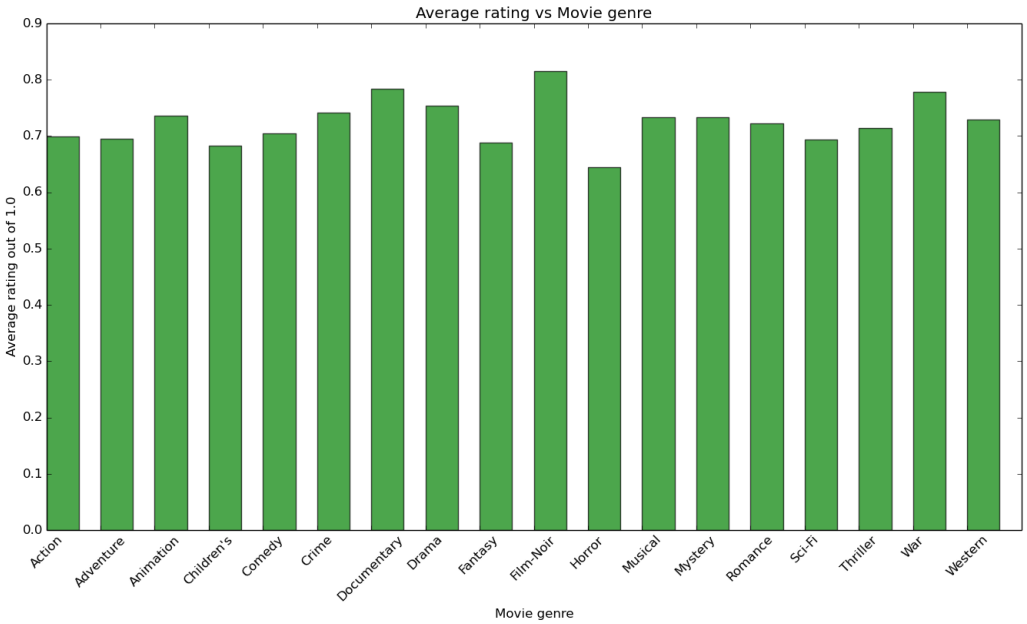

Figure 4: Average rating for various movie genres

3. MOVIE RATING PREDICTION

In this section we discuss the task of predicting the rating given a user and a movie. Given a user u and a movie m, we would like to predict the rating that u will give to m. Although this problem is closely related to the problem of recommending the best possible items to a user based on the user's previous purchases or reviews, in this work we focus on accurately predicting the rating itself. Accordingly, we evaluate the performance of our models using the mean-squared-error (MSE) [17] and the coefficient of determination R^2 [15] metrics. These are calculated as follows:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(f(u_i, m_i) - r(u_i, m_i)\right)^2$$

$$R^2 = 1 - \frac{MSE}{\text{variance in true ratings}}$$

where N is the number of examples, u_i is the user in the example i, m_i is the movie in the example i, $f(u_i, m_i)$ is the predicted rating, $r(u_i, m_i)$ is the true rating.

The MSE is a useful metric for comparing 2 or more models on the same data. However, the MSE depends on the variance in the data and hence by itself is not a good measure of how well a model explains the data. The R^2 value, on the other hand, is a measure of the fraction of the variance that is explained by a model, and hence is a better measure of how well a model explains the data. We report both values for each model we consider.

As a baseline model, we consider a linear predictor based on user and item features, which we discuss in section 6.

Further, to confirm the validity of our results, we compare them with the results obtained using the Apache Mahout machine learning library [1] on the same data set. Apache Mahout contains an implementation of an alternating-least-squares recommender with weighted-λ-regularization (ALS-WR) [28, 8].

4. RELATED WORK

The MovieLens data set [6, 7] is a data set collected and made available by the GroupLens Research group [5]. MovieLens is a website for personalized movie recommendations [10]. The data set contains data from users who joined MovieLens in the year 2000.

Given a set of users, items and ratings given by some of the users for some of the items, the task of predicting the rating for a given user for a given movie has been studied in the literature in the form of *recommender systems*, which additionally perform the task of recommending items to users. The state-of-the-art in recommender systems is based on 2 main approaches - *neighborhood approach* and *latent factor models* [21, 23]. Both these methods rely only on past user behavior, without using any features about the users or items. [28] describes an Alternating-Least-Squares with Weighted-λ-Regularization (ALS-WR) parallel algorithm designed for the Netflix Prize challenge [11], a well-known competition for the best algorithm to predict user ratings for films, given only the previous ratings, without any other information about the users or films. An alternative to these methods is *content filtering* [23], which uses user features (such as age, gender, etc.) and item features (such as movie genre, cast etc.) instead of relying on past user behavior.

Neighborhood approaches and latent factor models are generally more accurate than content-based techniques [23], since they are domain-free and capture subtle information (such as a user's preferences) which is hard to extract using content-filtering. On the other hand, neighborhood approaches and latent factor models suffer from the *cold start* problem i.e. they are not useful on new users or new items [13].

In this work, we compare 3 models - the first one is a content-based linear predictor and is described in section 6. The second one is a neighborhood model and is described in section 7. The last one is a latent-factor model and is described in section 8.

5. FEATURES

Sections 6, 7, 8 discuss the 3 models we consider, namely linear regression, collaborative filtering and latent factor[23, 22]. While collaborative filtering and latent factor models do not use the additional information about users and items available in the dataset, the model based on linear regression relies on this information. As discussed in section 2, the user's age and occupation and the movie genre seem to be useful in predicting the rating that a user will give to a movie.

The data set includes the user's age as one of 6 values - 1, 18, 25, 35, 45, 50, 56, which respectively denote the age groups under 18, 18-24, 25-34, 35-44, 45-49, 50-55, above 56. During the pre-processing stage, we replace each of these values by an integer in the interval $[0, 6]$. When we create the feature vector for a given user-movie pair, we create one boolean feature for each age group, where a value of 1 indicates that the user falls in the corresponding age group. Thus, we have 7 features to represent the user's age.

The data set includes the user's occupation as an integer in the interval $[0, 20]$, where each integer denotes a particular occupation as given in [7]. When we create the feature vector for a given user-movie pair, we create one boolean feature for each occupation, where a value of 1 indicates that the user's occupation is the corresponding occupation. Thus, we have 21 features to represent the user's occupation.

The data set includes the genres for a movie as a string of genres separated by '|'. Each genre in this string is one of 18 possible genres given in [7]. We represent each genre as an integer in the interval $[0, 17]$. When we create the feature vector for a given user-movie pair, we create one boolean feature for each genre, where a value of 1 indicates that the corresponding genre is one of the genres listed for that movie. Thus, we have 18 features to represent the genres of the movie.

Similarly, we represent the user's gender by 2 boolean features. Although the user's gender does not seem to be useful, we still include it in the features used by the predictor and

expect that the the predictor will learn a weight close to 0 for these 2 features.

We note that though the features just described capture useful attributes of users and movies, none of these features capture a sense of compatibility or a match between a user and a movie. In order to capture this, we define an additional feature as follows. For a user u, define a vector u_g, where u_{gi} is the average rating given by the user for movies which include the genre i i.e. the vector u_g is the vector of average ratings given by the user for all the genres. The length of this vector is $G = 18$ where is the G is the number of distinct movie genres in the data set. For a movie m, we define a vector m_g which is just the vector of boolean values for all genres, denoting the genres listed for that movie. Then, we compute the dot product of vectors u_g and m_g and divide it by the number of genres n_g listed for the movie. i.e. we compute the value $(u_g \cdot m_g)/n_g$ and use it as a feature in the linear predictor. The intuition is that the term $u_g \cdot m_g$ tries to capture the match between the user's preferences for the specific genres which have been listed for that movie. Dividing by the number of genres listed for that movie makes sure that the case where the user's average rating is high for only one of the genres listed for that movie is given less weight compared to the case where the user's average rating is high for *all* or most of the genres listed for that movie.

We introduce 3 additional features: the average rating given by user u, the average rating for the movie m and a constant feature (set to 1) for the intercept. Thus, the total number of features we use is 7+21+18+2+1+3 = 52. We describe the linear predictor using these features in the following section.

6. LINEAR PREDICTOR

Using the features described in the previous section, we can write a linear model as follows:

$$f(u, m) = X_{um} \cdot \theta$$

where X_{um} is the feature vector of length 52 representing the user-movie pair (u, m) and θ is the set of weights to be learned.

The error function is defined as follows:

$$F(\theta) = \frac{1}{N} \sum_{i=1}^{N} (r(u_i, m_i) - X_i \cdot \theta)^2 + \lambda \sum_{i=2}^{M} \theta_i^2$$

where $M = 52$ is the number of features, λ is the regularization hyper-parameter, X_i is the feature vector representing the i'th training example. The term $\sum_{i=2}^{M} \theta_i^2$ penalizes model complexity and reduces overfitting [14]. Note that we do not regularize the intercept term. λ is used to control the trade-off between accuracy and complexity during training.

Then we find the θ which minimizes the error function, as follows

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} (r(u_i, m_i) - X_i \cdot \theta)^2 + \lambda \sum_{i=2}^{M} \theta_i^2$$

We have the following expressions for the gradient of the error function with respect to θ:

$$\frac{\partial F}{\partial \theta_1} = \frac{1}{N} \sum_{i=1}^{N} -2(r(u_i, m_i) - X_i \cdot \theta)$$

$$\frac{\partial F}{\partial \theta_k} = \frac{1}{N} \sum_{i=1}^{N} 2(r(u_i, m_i) - X_i \cdot \theta)(-X_{ik}) + 2\lambda\theta_k$$

for $2 \leq k \leq M$

We find $\hat{\theta}$ using the implementation of the L-BFGS algorithm[19] included in the SciPy library [20]. Computing the error function and its gradient with respect to θ is the computational bottleneck during training since the entire training set with around 640K examples needs to be processed. In order to speed up this computation, we use Cython with Numpy [12, 3]. As a result, the entire run consisting of training as well as making predictions takes about 8 minutes on a conventional laptop.

We tune the regularization hyper-parameter λ by performing a line-search over the values (0.0001, 0.001, 0.01, 0.1, ...) and choosing the value which gives the highest value of R^2 on the validation set. We observe that $\lambda = 0.001$ gives the best results.

To confirm that our gradient computation is correct, we compute the ratio of the magnitude of the gradient computed numerically using SciPy and the magnitude of the gradient returned by our implementation. We observe that the ratio is very close to 1.0, confirming that our gradient computation is correct.

7. COLLABORATIVE FILTERING

The next model we consider is a collaborative filtering algorithm using an item-oriented approach [24, 27]. We use the *adjusted cosine similarity* metric to compute similarity between items i and j as follows [25]:

$$sim(i, j) = \frac{\sum_{u \in U} (R_{u,i} - \bar{R_u})(R_{u,j} - \bar{R_u})}{\sqrt{\sum_{u \in U} (R_{u,i} - \bar{R_u})^2} \sqrt{\sum_{u \in U} (R_{u,j} - \bar{R_u})^2}}$$

Here $\bar{R_u}$ is the average of the u'th user's ratings, $R_{u,i}$ and $R_{u,j}$ are the ratings given by user u to items i and j respectively. U is the set of users which have rated both item i and item j.

Then, we compute the prediction on an item i for a user u by computing the weighted sum of the ratings given by the user on the items similar to i [26].

$$P_{u,i} = \frac{\sum_{\text{all similar items},N}(sim(i,N)R_{u,N})}{\sum_{\text{all similar items, N}}(|sim(i,N)|)}$$

Here, we can consider the top K most similar items to the given item i [16]. We observe that setting K to its maximum possible value, which is the number of items other than i rated by user u, gives the best results in terms of the R^2 value.

The computation of similarity values between items and the subsequent prediction of ratings using them are the computational bottlenecks involved in scaling the model to a dataset of size 1 million ratings. Hence, we implement this model using C++ instead of using a higher level language. As a result, the entire run over the training, validation and test sets takes about 5 minutes on a conventional laptop.

8. LATENT-FACTOR

The next model we consider is a latent factor model [23]. Given a user u and a movie i, we predict the rating that the user will give to the movie as follows:

$$f(u,i) = \alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i$$

where α is a global bias, β_u and β_i are user and movie biases respectively. γ_u and γ_i are latent factors for user u and movie m respectively, which will be learned during the training process. γ_u and γ_i are K-dimensional vectors.

The error function F is defined as

$$F(\Theta) = \frac{1}{N}\sum_{u,i}(f(u,i) - R_{u,i})^2 + \lambda(\sum_u \beta_u^2 + \sum_i \beta_i^2 + \sum_u \|\gamma_u\|_2^2 + \sum_i \|\gamma_i\|_2^2) \quad (1)$$

where $\Theta = (\alpha, \beta_u, \beta_i, \gamma_u, \gamma_i)$ for $1 \le u \le U$ and $1 \le i \le I$, where U is the number of distinct users and I is the number of distinct movies. The term $\sum_u \beta_u^2 + \sum_i \beta_i^2 + \sum_u \|\gamma_u\|_2^2 + \sum_i \|\gamma_i\|_2^2$ penalizes model complexity and reduces overfitting [13]. Note that we do not regularize the global bias α. λ is used to control the trade-off between accuracy and complexity during training.

We find parameters $\hat{\Theta}$ which minimize the error function F as follows:

$$\hat{\Theta} = \arg\min_{\Theta} \frac{1}{N}\sum_{u,i}(f(u,i) - R_{u,i})^2 + \lambda(\sum_u \beta_u^2 + \sum_i \beta_i^2 + \sum_u \|\gamma_u\|_2^2 + \sum_i \|\gamma_i\|_2^2) \quad (2)$$

We have the following expressions for the gradient of the error function with respect to Θ:

$$\frac{\partial F}{\partial \alpha} = \frac{1}{N}\sum_{u,i} 2(f(u,i) - R(u,i))$$

$$\frac{\partial F}{\partial \beta_u} = \frac{1}{N}\sum_{i} 2(f(u,i) - R(u,i)) + 2\lambda\beta_u$$

$$\frac{\partial F}{\partial \beta_i} = \frac{1}{N}\sum_{u} 2(f(u,i) - R(u,i)) + 2\lambda\beta_i$$

$$\frac{\partial F}{\partial \gamma_{uk}} = \frac{1}{N}\sum_{i} 2(f(u,i) - R(u,i))\gamma_{ik} + 2\lambda\gamma_{uk}$$

$$\frac{\partial F}{\partial \gamma_{ik}} = \frac{1}{N}\sum_{i} 2(f(u,i) - R(u,i))\gamma_{uk} + 2\lambda\gamma_{ik}$$

As mentioned in [23], there are 2 approaches to solving the above optimization problem to find Θ - stochastic gradient descent and alternating least squares (ALS). We use stochastic gradient descent. At the beginning of each iteration, we randomly sort the training examples and use each example to update parameters. We run the algorithm for 1000 iterations (where each iteration is one pass through the entire training set). We observe that the R^2 value does not increase significantly beyond 1000 iterations. In order to enable the implementation to a large data set we implement the stochastic gradient descent using C++. As a result, the entire run completes in under 5 minutes.

We tune the dimensionality K of the latent factors and the regularization hyper-parameter λ by selecting values which maximize the R^2 on the validation set. We observe that the values $K = 15$ and $\lambda = 0.01$ give the best results.

To confirm that our gradient computation is correct, we compute the ratio of the magnitude of the gradient computed numerically using SciPy and the magnitude of the gradient returned by our implementation. We observe that the ratio is very close to 1.0, confirming that our gradient computation is correct.

9. COMPARISON OF THE MODELS

The linear predictor discussed in section 6 is a feature-based predictor since it uses user features (age, gender, occupation, etc.) and movie features (genre). In general, feature-based predictors for rating prediction suffer from the problem of not having enough features at the granularity required to capture individual tastes and subtle differences between items and between users [9]. On the other hand, user and item features are useful when there is not enough data per user and per item, or for new users and new items, also known as the cold-start problem [13].

The collaborative filtering model discussed in section 7 is known as a *neighborhood method* in literature [21]. Latent factor models are more expressive and tend to provide more accurate results than neighborhood models [21]. Latent factor models have the ability to capture and learn the latent factors which encode users' preferences and items' characteristics. On the other hand, neighborhood models are relatively simpler and offer more intuitive explanations of the

reasoning behind recommendations [21]. Both models suffer from the cold-start problem, but give good performance for users and items with enough data [9].

As mentioned in section 2, the MovieLens data set has enough data per user and per movie. Hence, the cold-start problem will not be a concern and we expect the collaborative filtering and latent factor models to perform better than the linear predictor.

10. RESULTS AND CONCLUSIONS

This section discusses the results of the 3 models discussed in the previous sections. We divide the data set of around 1M ratings using a 80%-20% split and use the latter as the test set. The former is further divided using a 80%-20% split to get the training and validation sets respectively. We run each of the models on the same training, validation and test sets for a fair comparison. Table 1 lists the MSE and the R^2 values (bold-faced and in brackets) for each of the models.

In order to confirm the validity of our results, we compare them with those obtained using Apache Mahout[1], a machine learning library. Apache Mahout contains an implementation of an alternating-least-squares recommender with weighted-λ-regularization (ALS-WR) [28, 8]. The Mahout package contains an example [2] of using Mahout commands to experiment with a recommender system on the same MovieLens-1M [6] dataset. The original example [2] in the Mahout package takes the entire ratings file as the input, creates a training set and a test set using a 90-10 % split, runs parallel alternating least squares, computes predictions and recommendations, and reports the Root-Mean-Square-Error (RMSE) [18]. To enable a fair comparison, we modify the example in order to make it use the same training, validation and test sets we use to evaluate our implementations. Further, the example uses the input data as is, without normalization of the ratings, and hence produces output ratings in the range 0-5 (we confirm this by printing out the predicted ratings). The example uses specific values for hyper-parameters such as the regularization hyper-parameter λ, dimensionality K of latent factors etc. which have been tuned for the original data set without normalization. Hence, in order to get the best possible results from the Mahout example, we do not normalize the data before giving it as an input to the example. Since the RMSE reported by the example is for the original unnormalized data, we separately compute the variance of the original unnormalized ratings, and use it to compute the R^2 value obtained by the example. Thus, we compare our results with those obtained using the Mahout example in terms of the R^2 value. Table 1 lists only the R^2 value for the Mahout column.

We observe that our results with the latent factor model are quite close to those obtained using Mahout's recommender. This is expected, since Mahout's ALS recommender itself is based on a latent factor model [28, 8]. This confirms the validity of our results.

The results in table 1 indicate that the latent factor model performs the best among the 3 models we consider. This is expected, as mentioned in section 9. We also observe that the linear predictor performs worse compared to the other models on the training set. This is also expected, as mentioned in section 9. However, it gives results comparable to collaborative filtering on the validation set.

We now analyze the values of the learned model parameters θ of the linear predictor. Figure 5 plots the value of θ_i for all $0 \leq i < 52$. We observe that the model is dominated by the first 4 parameter values which correspond to the constant intercept feature, average user rating for the given user, average item rating for the given item, and the value $(u_g \cdot m_g)/n_g$ as described in section 6. The values indicate that the rating depends much more on these 4 features compared to the remaining features. Further, note that the 2nd, 3rd and 4th of these features directly attempt to characterize users and movies in terms of *ratings*, as opposed to remaining features, which characterize users and movie only in terms of their attributes, without considering any of the ratings. This also supports the observation that neighborhood models and latent factor models give more accurate results than content-based approaches [23], since neighborhood models and latent factor models attempt to describe users and items based on past ratings, instead of relying on their attributes.

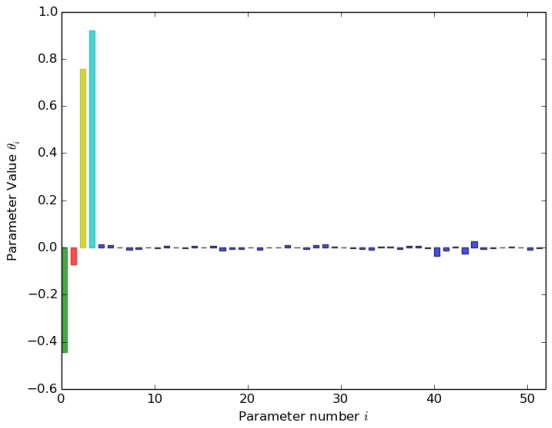

Figure 5: All parameter values of the trained linear predictor discussed in section 6

Based on the results, we conclude that latent factor models tend to perform better than neighborhood approaches and content-based approaches on the movie rating prediction task, provided there is enough data for each user and for each movie.

11. RESULTS ON GOOGLE DATA SET

We also evaluate the latent factor model described in 8 on a data set of restaurant ratings extracted from the Google Local data set [4]. We extract ratings of restaurants based on the categories given in the ratings. There are about 4M restaurant ratings, which we split into parts of sizes 2M, 1M and 1M for the training, validation and test sets respectively respectively. The training set of 2M ratings contains ratings for about 1.1M unique users and 667K unique places. Thus, there are about 2 ratings per user on an average and about 3 ratings per place on an average, in contrast to the MovieLens which has many more ratings per user and per movie.

	# Examples	Variance	Linear Predictor	Collaborative Filtering	Latent Factor	Mahout ALS
Training	640135	0.049970	0.029748 **(0.404688)**	0.023681 **(0.526092)**	0.020354 **(0.592680)**	**(0.586866)**
Validation	160033	0.049826	0.033429 **(0.329085)**	0.033488 **(0.327886)**	0.030545 **(0.386954)**	**(0.392062)**
Test	200041	0.049818	0.033776 **(0.322010)**	0.033779 **(0.321948)**	0.030765 **(0.382451)**	**(0.388799)**

Table 1: MSE and R^2 obtained using the 3 predictors discussed in this work and Mahout's ALS recommender on the MovieLens dataset. Values in boldface/brackets are R^2 values.

The latent factor model discussed in section 8 gives very high R^2 values on the training set and very low R^2 values on the validation set. This clearly indicates drastic overfitting on the training data. Increasing the value of the regularization hyper-parameter decreases the R^2 value on the training set but does not significantly improve results on the validation set.

This result highlights the limitation of latent factor models on sparse data with very few ratings per user and per item.

12. REFERENCES

[1] Apache Mahout. http://mahout.apache.org/.

[2] Apache Mahout MovieLens 1M example. https://github.com/apache/mahout/blob/master/examples/bin/factorize-movielens-1M.sh.

[3] Cython Tutorials - Working with NumPy. http://docs.cython.org/src/tutorial/numpy.html.

[4] Google Local dataset. http://jmcauley.ucsd.edu/data/googlelocal.tar.gz.

[5] GroupLens. http://grouplens.org/.

[6] GroupLens MovieLens 1M dataset. http://grouplens.org/datasets/movielens/.

[7] GroupLens MovieLens 1M dataset. http://files.grouplens.org/datasets/movielens/ml-1m-README.txt.

[8] Introduction to ALS Recommendations with Hadoop. https://mahout.apache.org/users/recommender/intro-als-hadoop.html.

[9] Latent Factor Models for Web Recommender Systems. http://www.ideal.ece.utexas.edu/seminar/LatentFactorModels.pdf.

[10] MovieLens. https://movielens.org/.

[11] Netflix Prize. http://en.wikipedia.org/wiki/Netflix_Prize.

[12] NumPy. http://www.numpy.org/.

[13] Recommender Systems. http://cseweb.ucsd.edu/~jmcauley/cse255/slides/lecture5_recommender.pdf.

[14] Supervised Learning - Regression. http://cseweb.ucsd.edu/~jmcauley/cse255/slides/lecture1_supervised.pdf.

[15] Wikipedia - coefficient of determination. http://en.wikipedia.org/wiki/Coefficient_of_determination.

[16] Wikipedia - collaborative filtering. http://en.wikipedia.org/wiki/Collaborative_filtering.

[17] Wikipedia - mean squared error. http://en.wikipedia.org/wiki/Mean_squared_error.

[18] Wikipedia - root-mean-square-error. http://en.wikipedia.org/wiki/Root-mean-square_deviation.

[19] R. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific and Statistical Computing*, 16, September 1995.

[20] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.

[21] Y. Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4, January 2010.

[22] Y. Koren and R. Bell. Advances in collaborative filtering. *Recommender Systems Handbook*, 2011.

[23] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42:30–37, August 2009.

[24] G. Linden, B. Smith, and J. York. Amazon.com recommendations item-to-item collaborative filtering.

[25] B. Sarwar. Item-based collaborative filtering algorithm - adjusted cosine similarity. http://www10.org/cdrom/papers/519/node14.html.

[26] B. Sarwar. Item-based collaborative filtering algorithm - prediction computation - weighted sum. http://www10.org/cdrom/papers/519/node16.html.

[27] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms.

[28] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-scale parallel collaborative filtering for the netflix prize.