MOVIE RECOMMENDATION - A COMPARISON OF FOUR DIFFERENT MODELS[*]

AMIRHOSSEIN ALVANDI, JOHN MILMORE, ZHOU TANG, YUJIAN WU[†]

Abstract. In this project, we implement the recommender systems with four different kinds of algorithms, which are memory-based KNN, matrix factorization, factorization machine and a probabilistic neural network. By applying the models to a dataset with 2352 users' 31620 ratings on 1465 movies, we get the MSE for the fours models, which are 1.12, 0.87, 0.88 and 0.62 respectively. Therefore, the deep learning method outperforms the factorization method which is better than the memory-based KNN model.

Key words. Recommendation, Collaborative Filtering, Memory based KNN, Matrix factorization, Deep learning

AMS subject classifications. 68Q25, 68R10, 68U05

1. Introduction.

Broadly speaking, recommender systems are based on one of two strategies. The content filtering approach which creates a profile for each user or product to characterize its nature. For example, a movie profile could include attributes regarding its genre, the participating actors, and so forth. Also, user profiles might include demographic information such as the user's age, gender, etc. The profiles allow programs to associate users with matching movies. Of course, content-based strategies require gathering external information that might not be available or easy to collect.

On the other hand collaborative filtering method relies only on past user behavior. For example, previous ratings that they gave to different movies without requiring the creation of explicit profiles. While generally more accurate than content-based techniques, collaborative filtering suffers from what is called the cold start problem, due to its inability to address the system's new products and users[**?**]. The two primary areas of collaborative filtering are the neighborhood methods and latent factor models.

The idea behind neighborhood collaborative filtering methods are very intuitive. A prediction is made for the rating of a user on a movie by considering the rating given to similar movies by that user. This is called item-based collaborative filtering and is widely used in movie recommendation systems. A variant of this is called user-based collaborative filtering. The difference here is that ratings are predicted by considering the ratings of similar users on the movie of interest. Both of these approaches can be combined to account for instances where users have not rated many movies or movies have not been rated by many users.

In latent factor approach the ratings are explained by characterizing both movies and users. To this end matrix factorization is one of the most applied techniques that allows us to embed high dimensional data in lower dimensional spaces which mitigate effects due to noise, uncover latent relations, and facilitates further processing. In its basic form, matrix factorization characterizes both items and users by vectors of factors inferred from movie rating patterns. Accordingly, it provides recommendations based on correspondence level between movie and user factors. This method became widely known during the Netflix prize challenge due to its effectiveness in dimensionality reduction and sparsity removal.

[*] Submitted to the editors DATE.

[†] Department of Mathematics and Statistics, UMass Amherst, ID

As a sub-field of machine learning, deep learning has gained tremendous success in many tasks, especially with the improvement of neural networks [?]. With the good properties of flexibility, non-linear transformation, latent variable modeling, different kinds of deep neural networks were developed to deal with various kinds of recommendation tasks. For instance, multilayer perceptron (MLP) was used to model the non-linearity between the users and items[?]; convolutional neural network was used to extract textual and visual information for image or video recommendation[?]; recurrent neural network was successful in the news or review texts recommendation because of the ability of capturing users' sequential behavior pattern[?]. In our project, we use a MLP based neural network to predict a user's ratings on the movies. But instead of outputting merely the ratings, we have modified the neural network so it could output gaussian distributions, which would give us the prediction uncertainty.

The paper is organized as follows: Our methods are in section 2, experimental results are in section 3, discussion is in section 4 and the conclusions follow in section 5.

2. Methodology.

Here we discuss, in detail, the implementation of each model considered in this project.

2.1. Neighborhood CF.

As mentioned earlier, in collaborative filtering the idea is to filter information items and remove those that are not relevant to the user or item we would like to make predictions for [?]. One way to do this is to consider only the top-k candidates based on a similarity metric.

2.1.1. Item-based KNN.

The idea behind item-based k-nearest neighbor collaborative filtering is as follows: in order to determine the rating of user u on movie m, we can find other movies that are similar to movie m, and based on user u's ratings on those similar movies we infer his rating on movie m [?]. The idea here is very intuitive. We want to predict the rating for a given user on a given movie. To do so, we look at all the movies that the user has rated. Then, we consider only the k movies calculated as being the most similar to the movie of interest. We then base our prediction on the user's ratings of these similar movies. The calculation of a prediction of user u on movie m is shown below.

$$P_{m,u} = \frac{\sum_{m' \in N_u^K(m)} R_{m',u} sim(m,m')}{\sum_{m' \in N_u^K(m)} |sim(m,m')|} \tag{2.1}$$

where $N_u^K(m)$ is equal to $\{m' : m'$ has been rated by u and is in the top k most similar movies to $m\}$. We can see that the prediction is a weighted average of ratings weighted by similarity. In order to determine the similary between movies we must define a similarity metric. One option would be the cosine similarity metric:

$$sim(m_1,m_2) = \frac{\sum_{u \in U(m_1,m_2)} (R_{m_1,u})(R_{m_2,u})}{\sum_{u \in U(m_1,m_2)} (R_{m_1,u})^2 \sum_{u \in U(m_1,m_2)} (R_{m_2,u})^2} \tag{2.2}$$

where $U(m_1, m_2)$ is equal to $\{u : u$ has rated both m_1 and $m_2\}$. Although this is a valid similarity metric, it is flawed in the domain of movie recommendations. This calculation does not take into account the different rating schemes of different users. For example, some users may be strict critics and never rate movies above a 3. On the other hand, some users may love every movie they see and never rate a movie below a 3. Taking this into consideration we use the adjusted cosine similarity metric:

$$sim(m_1, m_2) = \frac{\sum_{u \in U(m_1,m_2)} (R_{m_1,u} - \bar{R_u})(R_{m_2,u} - \bar{R_u})}{\sum_{u \in U(m_1,m_2)} (R_{m_1,u} - \bar{R_u})^2 \sum_{u \in U(m_1,m_2)} (R_{m_2,u} - \bar{R_u})^2} \tag{2.3}$$

Here we adjust each movie rating by the average rating of the user.

2.1.2. User-based KNN. Another, similar approach to item-based KNN CF is the user-based alternative. This is essentially the same algorithm, but instead of basing our prediction on ratings given by the user on similar movies, we consider the ratings given by similar users on the movie of interest. This is also a very intuitive approach. For a given user u and movie m that we would like to predict the rating for, we consider only the ratings given by the k users most similar to u on m. We then make our prediction by taking the weighted average of these ratings, weighted by similarity. The prediction is calculated as:

$$P_{m,u} = \frac{\sum_{u' \in N_m^K(u)} R_{m,u'} sim(u, u')}{\sum_{u' \in N_m^K(u)} |sim(u, u')|} \tag{2.4}$$

where $N_m^K(u)$ is equal to $\{u' : u'$ has rated m and is in the top k most similar users to $u\}$. The similarity between users is calculated in a similar manner as in item-based:

$$sim(u_1, u_2) = \frac{\sum_{m \in M(u_1,u_2)} (R_{m,u_1} - \bar{R_m})(R_{m,u_2} - \bar{R_m})}{\sum_{m \in M(u_1,u_2)} (R_{m,u_1} - \bar{R_m})^2 \sum_{m \in M(u_1,u_2)} (R_{m,u_2} - \bar{R_m})^2} \tag{2.5}$$

where $M(u_1, u_2)$ is equal to $\{m : m$ has been rated by both u_1 and $u_2\}$. Here we are calculating the similarity between two users. In this case we would like to compare the ratings on movies that both users have rated. Note how we adjust the ratings as we did in the item-based method, but now we do so in terms of the average movie ratings. This is to account for the various movie rating distributions. For example, some movies may be highly rated across the board whereas others may have an even distribution of ratings.

2.1.3. User/Movie-based KNN. A third approach that was considered was a combination of item-based and user-based KNN CF. In this approach we considered both the ratings on the k most similar movies to m by u and the ratings of the k most similar users to u on m. The similarity between movies is calculated as in 2.3 and the similarity between users is calculated as in 2.5. The predicted rating is calculated as the following:

$$P_{m,u} = \frac{\sum_{u' \in N_m^K(u)} R_{m,u'} sim(u, u') + \sum_{m' \in N_u^K(m)} R_{m',u} sim(m, m')}{\sum_{u' \in N_m^K(u)} |sim(u, u')| + \sum_{m' \in N_u^K(m)} |sim(m, m')|} \tag{2.6}$$

This approach gives the best results compared to just item-based or just user-based as shown in the results.

2.2. Matrix Factorization. Suppose we want to approximate the sparse user-movie rating matrix R as the product of two matrices

$$R_{|U| \times |D|} \approx \hat{R}_{|U| \times |D|} = P_{|U| \times |K|} Q^T_{|K| \times |D|}$$

where $|U|$, and $|D|$ refer to the number of users, and the number of movies respectively, and K is the number of latent factors that we want to discover. In this way, each row of P would represent the strength of the association between a user and the features. Similarly, each row of Q stands for the the strength of association between a movie and the features.

Earlier recommendation systems relied on imputation to fill in missing ratings and make the rating matrix dense. However, imputation can be very expensive as it significantly increases the amount of data. In addition, inaccurate imputation might distort the data considerably. Hence, more recent works suggested modeling directly the observed ratings only.

To learn the factor vectors (rows of P, and columns of Q^T), the system minimizes the regularized squared error on the set of known ratings

$$e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj})^2 + \frac{\beta}{2} \sum_{k=1}^{K} (||P||^2 + ||Q||^2) \tag{2.7}$$

It is important to note that our goal is to generalize the previous ratings to predict future, unknown ratings. Thus, the system should avoid overfitting the observed data by regularizing the learned parameters, whose magnitudes are penalized. The constant β controls the extent of regularization and is usually determined by cross-validation.

2.2.1. Stochastic Gradient Descent. In order to minimize the prediction error iteratively, Simon Funk popularized a stochastic gradient descent optimization of Equation (2.7) wherein the algorithm loops through all ratings in the training set. Then it modifies the parameters by a magnitude proportional to α in the opposite direction of the gradient. Having obtained the gradient, we are now able to formulate the update rules for both p_{ik}, and q_{kj} as follows

$$p_{ik}^{'} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\alpha (e_{ij} q_{kj} - \beta p_{ik})$$

$$q_{kj}^{'} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + 2\alpha (e_{ij} p_{ik} - \beta q_{kj})$$

Here, α is the learning rate which heavily influence the convergence of the algorithm. Large values of α makes the algorithm to take huge steps, and it might jump across the local minimum point and thereby missing it.

2.2.2. Implementing Bias. Regarding the nature of rating data, it is also important to consider how these ratings are generated. One benefit of matrix factorization approach to collaborative filtering is its flexibility in dealing with various additional sources of information about both the users and the movies. For instance, it is more accurate to include some biases contributing to the observed ratings. For example there are users who may generally tend to give lower or higher ratings than other users. Also, there are certain movies that are more likely to get higher or lower ratings comparing to the other movies. Hence, to obtain the predicted ratings, we can also implement these biases as in [**?**] to better model the generated ratings

$$\hat{r}_{ij} = b + bu_i + bd_j + \sum_{k=1}^{K} p_{ik} q_{kj}$$

Here, b is the global bias which can be estimated by the mean of all given ratings, bu_i refers to the bias engaged with the i-th user, and bd_j represents the bias of the j-th movie. Then by applying the Stochastic Gradient Descent to minimize the bias in the same manner we discussed above, the update rules for the user and movie biases can be obtained as

$$bu_i^{'} = bu_i + \alpha \times (e_{ij} - \beta bu_i)$$
$$bd_j^{'} = bd_j + \alpha \times (e_{ij} - \beta bd_j)$$

It has been shown that by adding the bias in our algorithm, the convergence rate increases remarkably.

2.3. Factorization Machine. In the recommendation system, a linear or a logistic modeling technique is great and does well in a variety of problems. But the drawback is that the model only learns the effect of all variables or features individually rather than in combination. More specifically,

$$\begin{aligned} \text{rating} &= \beta_0 + \beta_1\text{age} + \beta_2\text{gender} + \beta_3\text{movieID} + \beta_4\text{year} + \beta_5\text{genre} \\ &= \beta_0 + \sum_{i=1}^{p} \beta_i x_i \end{aligned} \tag{2.8}$$

However, the interaction term in recommendation system is also important, e.g. gender and age. It's reasonable to think that kids love to see the comics, while the old man love to see fiction. So we can have another improved model.

$$\begin{aligned} \text{rating} &= \beta_0 + \beta_1\text{age} + \beta_2\text{gender} + \beta_3\text{movieID} + \beta_4\text{year} + \beta_5\text{genre} \\ &\qquad + \beta_{12}\text{age} \cdot \text{gender} + \cdots + \beta_{45}\text{year} \cdot \text{genre} \\ &= \beta_0 + \sum_{i=1}^{p} \beta_i x_i + \sum_{i=1}^{p-1} \sum_{j=i+1}^{p} \beta_{ij} x_i x_j \end{aligned} \tag{2.9}$$

The formula have already solved the interaction problem. However, it introduces another new problems. Due the sparisty of the dataset, the majority of the interaction terms are 0, it is hard for algorithm to optimzate it to get the estimate of the coefficient of the interaction term. At here, we introduce Matrix Factorization[**?**] to solve the sparisty problem.

The basic idea in here is:

$$\begin{bmatrix} \beta_{11} & \beta_{12} & \cdots & \beta_{1p} \\ \beta_{21} & \beta_{22} & \cdots & \beta_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{p1} & \beta_{p2} & \cdots & \beta_{pp} \end{bmatrix} = VV^T \tag{2.10}$$

where $\mathbf{V} \in \mathbb{R}^{p \times k}, k < p$ is a embedding matrix need to be trained. Hence the model equation for a factorization machine of degree $d = 2$ is defined as:

$$\begin{aligned} \text{rating} &= \beta_0 + \beta_1\text{age} + \beta_2\text{gender} + \beta_3\text{movieID} + \beta_4\text{year} + \beta_5\text{genre} \\ &\qquad + \beta_{12}\text{age} \cdot \text{gender} + \cdots + \beta_{45}\text{year} \cdot \text{genre} \\ &= \beta_0 + \sum_{i=1}^{p} \beta_i x_i + \sum_{i=1}^{p-1} \sum_{j=i+1}^{p} \langle \vec{v}_i, \vec{v}_j \rangle x_i x_j \end{aligned} \tag{2.11}$$

where the model parameters that have to be estimated are:

$$\beta_0 \in \mathbb{R},\ \vec{\beta} \in \mathbb{R}^p,\ \mathbf{V} \in \mathbb{R}^{p \times k} \tag{2.12}$$

And $\langle \cdot, \cdot \rangle$ is the dot product of two vector of size k;

$$\langle \vec{v}_i, \vec{v}_j \rangle = \sum_{f=1}^{k} v_{if} v_{jf} \tag{2.13}$$

A row v_i within V describes the i-th variable with k factors. $k \in N_0^+$ is a hyperparameter that defines the dimensionality of the factorization.

A 2-way FM (degree $d = 2$) captures all single and pairwise interactions between variables:

- β_0 is the global bias
- β_i models the strength of the $i - th$ variable
- $\beta_{ij} = \langle \vec{v}_i, \vec{v}_j \rangle$ models the interaction between the $i - th$ and $j - th$ variable. Instead of using an own model parameter $\beta_{i,j} \in \mathbb{R}$ for each interaction, the FM models the interaction by factorizing it. We will see later on, that this is the key point which allows high quality parameter estimates of higher-order interactions ($d \geq 2$) under sparsity.

From the formula we get, the time complexity of training one record is $O(kp^2)$, which is unaffordable when sample size is huge. After reformulating, we can prove that the time complexity can be reduced to $O(kp)$ for training one record. The model equation of a factorization machine can be computed in linear time O(kn)

proof: the pairwise interactions can be reformulated:

$$\begin{aligned}
&\sum_{i=1}^{p} \sum_{j=i+1}^{p} \langle \vec{v}_i, \vec{v}_j \rangle x_i x_j \\
=&\frac{1}{2} \sum_{i=1}^{p} \sum_{j=1}^{p} \langle \vec{v}_i, \vec{v}_j \rangle x_i x_j - \frac{1}{2} \sum_{i=1}^{n} \langle \vec{v}_i, \vec{v}_i \rangle x_i x_i \\
=&\frac{1}{2} \sum_{f=1}^{k} \left(\big(\sum_{i=1}^{p} v_{i,f} x_i\big) \big(\sum_{j=1}^{p} v_{j,f} x_j\big) - \sum_{i=1}^{p} v_{i,f}^2 x_i^2 \right) \\
=&\frac{1}{2} \sum_{f=1}^{k} \left(\big(\sum_{i=1}^{p} v_{i,f} x_i\big)^2 - \sum_{i=1}^{p} v_{i,f}^2 x_i^2 \right)
\end{aligned} \tag{2.14}$$

This equation has only linear complexity in both k and n, hence the computation is in $O(kn)$.

2.4. Probabilistic neural network.

The architecture of the probabilistic neural network is shown in Figure 1. The neural network takes categorical variables "userID" and "movieID" as inputs. Then "userID" will be mapped to a 150×1 random vector by a 2353×150 embedding layer, while "movieID" will be mapped to a 150×1 random vector by a 1465×150 embedding layer. Then the two vectors are concatenated to a 300×1 feature vector, which is passed to a dropout layer with a dropout rate of 0.02. Then the feature vector is fed to three fully connected (FC) layers, each of which has a dimension of 300×500, 500×500, 500×500. After each of the FC layers, we use ELU function as the non-linear activation function[?], and one dropout layer (dropout rate of 0.1) to prevent the model from overfitting. The neural network has two output layers, which would output the mean and log-variance of a gaussian distribution.

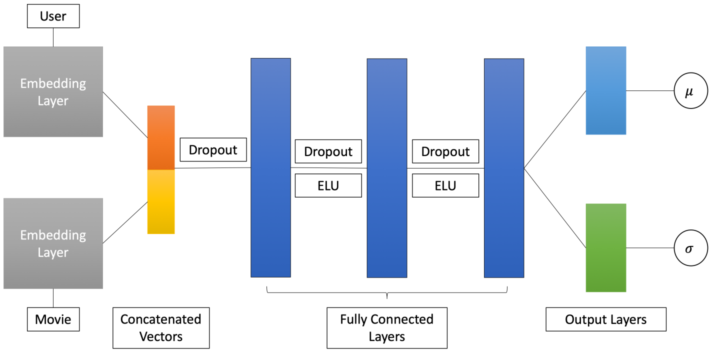

FIG. 1. *Architecture of the neural network.*

By assuming that each of the ratings has a gaussian distribution and that all the ratings are mutually independent, we didn't use the MSE as the loss function. Instead, we trained the model by maximizing the overall likelihood of the dataset, which is expressed as the product the likelihood of each of the ratings:

$$\underset{\mu_i,\ \sigma_i}{\text{argmax}}\ L(\mu_i, \sigma_i | x_i, y_i) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left[-\left(\frac{y_i - \mu_i}{\sigma_i}\right)^2\right]. \tag{2.15}$$

The neural network was trained with a ten-fold cross validation on the dataset, and we recorded both the log likelihood loss and MSE loss.

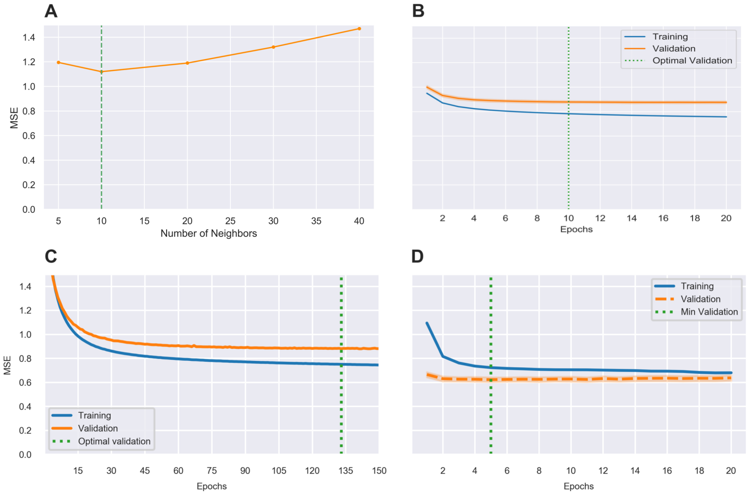

FIG. 2. *Training and validation loss of all four models.*

3. Experimental results. Figure 2 shows some results. Panel A shows the best performing neighborhood collaborative filtering implementation. This is representative of the combination of the item-based and user-based implementations. It can be seen that the optimal number of neighbors (similar movies and users) is 10 and this gives as MSE of about 1.12. As it can be observed from panel B, both training and validation MSE decrease substantially in the beginning epochs, and only after epoch 10 the validation MSE converges to its minimum(0.8722). Also, there is no sign of overfitting in the validation curve. Panel D reveals that as the training epoch increases, the average training MSE steadily decreases. But the average validation MSE decreases first, reaches the minimum (0.6221) at epoch 5, then goes up after epoch 5, which suggests the model is starting to get overfitting. Additional results are available in the supplement in Appendix A.

4. Discussion. Here we discuss the advantages and drawbacks related to each of the methods. The most interpretable model is the neighborhood based collaborative filtering. Although being easy to understand, the main drawback is its lack of scalability and its relatively low predictive accuracy. In addition to the relatively small prediction error of matrix factorization, this algorithm is easier to interpret comparing to its competitors. On the other hand, the main disadvantage of this technique is that it falls in the class of NP-hard optimization problems. Also, since this algorithm is not model-based, even by adding a single user, the algorithm is incapable of modeling it unless by retraining the whole model. The pros of the neural network is its high accuracy and uncertainty estimation. However, it's also the most complicated model. Besides, it's a black box algorithm, which has a poor interpretability.

5. Conclusions. Comparing the prediction error of the four methods, the neural network outperforms the matrix factorization methods which are more accurate than the memory based KNN.

Appendix A. Supplements.

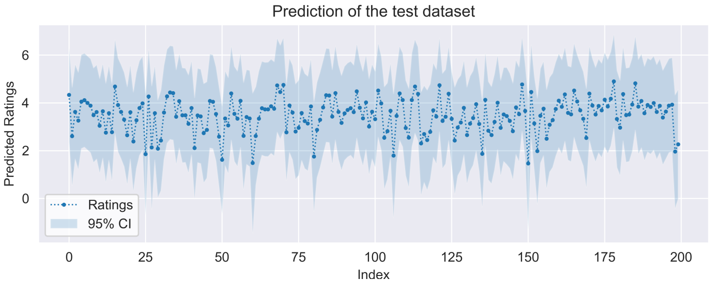

Fig. 3. *Prediction and the 95% confidence interval.*

Figure 3 shows the neural network's predictions of ratings for the test dataset. The dotted line is the predicted ratings while the light blue band is the 95% confidence interval constructed with the variance output by the neural network.

Acknowledgments. We would like to acknowledge the assistance of volunteers in putting together this example manuscript and supplement.