A Comprehnsive Analysis of Netflix Movie Recommendation System Implemented by Linear Regression and Restricted Boltzmann Machine

Dongyang Wu
School of Data Science
the Chinese University of Hong Kong, Shenzhen
Shenzhen, China
118010324@link.cuhk.edu.cn

Changwen Li
School of Data Science
the Chinese University of Hong Kong, Shenzhen
Shenzhen, China
118010134@link.cuhk.edu.cn

***Abstract*—This report is about the analysis and implementation of Netflix movie recommendation system. We have implemented two methods for Netflix movie recommendation system, which are LR (linear regression) and RBM (restricted Boltzmann machine). In this report, we also add some extensions to RBM, including momentum, adaptive learning rate, early stopping, regularization, mini-batch and biases. Besides all required RBM extensions, we also add another method to improve the performance of RBM, which is cross validation. The analysis and implementation of both linear regression and restricted Boltzmann machine will be introduced in this essay.**

Keywords—machine learning, recommendation system, linear regression, restricted Boltzmann machine

I. INTRODUCTION

Recommendation system is of great significance in real life. It is widely used in e-commerce. Recommendation system in Netflix, Amazon and Taobao has brought great convenience for customers and merchants. There are a lot of methods to implement recommendation system. In this essay, we make a research on the implementation of linear regression and restricted Boltzmann machine.

Linear regression is a classic machine learning algorithm based on supervised learning. It is used to predict the relationship between two variables by applying a linear equation to observed data. In this project, linear regression is implemented for rating prediction. To execute the program of linear regression, please run linearRegression.py.

A restricted Boltzmann machine (RBM) is a generative stochastic artificial neural network that can learn a probability distribution over its set of inputs. It has been proven to be efficient in many tasks, including dimension reduction, classification, feature learning, etc. In this project, we use RBM for movie recommendation. This essay introduces theory, implementation, analysis and discussion of RBM. To run the basic model with extensions, please run mainRBM.py. To run the cross validation RBM, please run runRBM.py. crossValidationLib.py and rbmClass.py provides some libraries used in cross validation.

II. THEORY

A. Linear Regression

When we implement a linear regression algorithm, we want to find the appropriate **b**, that

$$\min\|Ab - c\|^2$$

where **A** is a matrix that contains information from training data, and **c** is corresponding given results. $\|Ab - c\|^2$ could be written as:

$$(Ab - c)^T(Ab - c) = b^T A^T Ab - 2b^T A^T c + c^T c$$

Taking the derivative with respect to **b** and set it to 0, gives

$$2A^T Ab - 2A^T c = 0$$

Therefore, in order to minimize **b**, the corresponding b should be

$$(A^T A)^{-1} A^T c$$

Meanwhile, in case of overfitting, regularization should be implemented. Therefore, **b** is expected to be relatively small. So, the loss function is altered into:

$$\min\|Ab - c\|^2 + \lambda\|b\|^2$$

Again, taking the derivative with respect to **b** and set it to 0, gives **b** as

$$(A^T A + \lambda I)^{-1} A^T c$$

B. Retricted Boltzmann Machine

RBM is made of two layers which are the visible layers and the hidden layers. RBM is a bipartite graph where every node in hidden layer is connected to every node in visible layer. The weights of the connection is denoted as *W*, and the weight of connection between the *i*th node in visible layer and the *j*th node in hidden layer in hidden layer is denoted as W_{ij}. We denote a

vector *v* to represent the input (value of the visible layer) where v_i represents the *i*th value of visible layer. Here, v_i represents a vector of size 5 because the input is actually the rating of a user to a movie. Similarly, we denote h_j to represent the *j*th value of the hidden layer.

Now we focus on the inference of RBM. First, the value of visible layer will be passed to hidden layer through equation 1.

$$P(h_j = 1|v) = \sigma(W_{(ij)}^k v_i^k) \text{ (eq.1)}$$

Then, the value of the hidden layer will then be passed back to visible layer by equation 2.

$$P(v_i^k = 1|h) = softmax(\sum(h_j W_{(ij)}^k)) \text{ (eq.2)}$$

$P(v_i^k = 1|h)$ obtained from equation 2 demonstrates the possibility that movie *i* is rated *k* points by current user. To get the final rating of this movie, we can have two methods. We may choose the biggest value in vector v_i or average the value of each element in v_i by multiplying their scores (1, 2, 3, 4 or 5).

To train the model, the gradient descent techniques are used. In RBM, we will need to calculate the positive and negative gradients. To calculate positive gradient, equation 3 is used. To calculate negative gradient, equation 4 is used. The vector v in equation 3 is different from equation 4. The vector v in equation 4 is the negative data obtained from equation 2. Then we may use negative and positive gradient to implement gradient descent as shown in equation 5.

$$PG_{(ijk)} = P(h_j = 1|v)v_i^k \text{ (eq.3)}$$

$$NG_{(ijk)} = P(h_j = 1|v')v_i'^k \text{ (eq.4)}$$

$$W = W + learningRate * (PG - NG) \text{ (eq.5)}$$

To further improve the performance, we may use the techniques of momentum, biases, regularization, mini-batch, early stopping and adaptive learning rates. Momentum uses the past gradient and current gradient to update weight, which gives inertia to the model. To get momentum at time *t*, we simply add the former momentum at time *t-1* and the current gradient. Of course, we can adjust the proportion of the former momentum to adjust how much inertia we want in the new momentum. The equation of momentum is in equation 6.

$$Momentum_t = \alpha Momentum_{(t-1)} + (1 - \alpha)Gadient_t \text{ (eq.6)}$$

The regularization is used to punish the complicated and dense weight matrix. To achieve this, we calculate the L2 of weight matrix as its cost function. When update the weight matrix in training, we simply minus the multiplication of the matrix and regularization coefficient as show in equation 7.

$$W = W - \frac{\alpha}{2} W * learning\_rate \text{ (eq.7)}$$

Besides weight, to improve the performance, we also use the biases. Each node in visible and hidden layer has a bias value. To update the bias bv_i, we use equation 8. To update bias of hidden layer bh_i, we use equation 9.

$$bv = v - v' \text{ (eq. 8)}$$

$$bh = posHiddenProb - negHiddenProb \text{ (eq.9)}$$

The equation 1-4 also needs to be modified after the use of biases. The modification is quite simple and intuitive.

We also use early stopping and adaptive learning rate to prevent over-fitting. The idea is that we reduce the learning rate as RMSE decreases slowly or stop decreasing. We can also stop the training directly if RMSE decreases so slow or even stop decreasing. The implementation will be showed in the implementation part.

Mini-batch is used here to reduce the computing time. The implementation and analysis will be shown in the implementation and analysis part.

For better performance, we have implemented cross validation which is not required in the project description. The implementation and analysis will be discussed in discussion part.

III. Implementation

A. Linear Regression

In this project, A is a sparse matrix considered as the data of raters and movies, while c is altered from ratings. Each row of A represents a rate, where the elements representing the corresponding rater and movie set to 1 (e.g., in the first rating, the second people rated for the third movie; therefore, in the first row, the columns representing the second people or the third movie are set to 1, otherwise set to 0). Meanwhile, c is retrieved by subtracting the mean value of all rates (in the program, 'rBar') from original rates. Therefore, our prediction could be based on the average rating.

Meanwhile, restricted by actual rating, our final prediction will be restricted in [1,5]. If one prediction is larger than 5, it will be considered as 5; vice versa.

B. *Retricted Boltzmann Machine*

The fundamental part of the basic model is implemented by following the template. It is very simple and straight-forward. Therefore, we will not discuss this part here. We will discuss the implementation of adaptive learning rate, early stopping and mini-batch because they are much more difficult to implement.

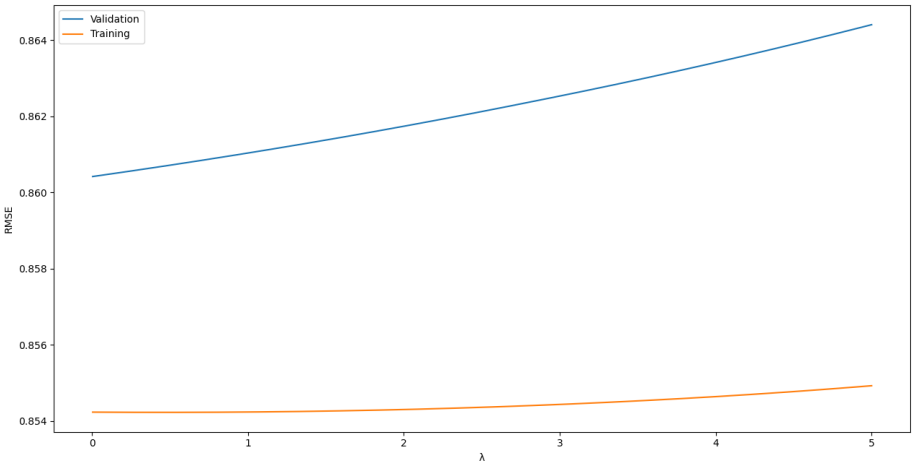
To implement adaptive stopping, we decrease learning rate by half or stop training when RMSE decreases too slow. To achieve this, we can compare the average value of certain number of RMSEs and the average value of another number of RMSEs. For instance, we may compare the average value of last 6 RMSEs and the average value of the last 9 RMSEs. If the difference between the two average value is smaller than a threshold such as 0.001, then we decrease the learning rate by half or directly stop training.

To implement mini-batch, we simply update the weight matrix only when the batch size number of users have been processed. To achieve this, a intuitive method is to set a counter to count how many users the machine has processed. After the counter exceeds the batch size, we do the gradient descent and reset the counter.

IV. ANALYSIS

A. *Linear Regression*

1001 points are evenly taken in the interval [0,5]. The RMSE graph excluding x=0 is shown as follows:



The minimum RMSE is 0.8604, when λ=0.005. While for λ=0, the RMSE for validation and training are 0.9974 and 0.9985 respectively. It could be seen that, with regularization, even a small λ could bring about critical changes of RMSE. Then the RMSE will slightly increase along with λ.

B. *Retricted Boltzmann Machine*

1) Analysis on number of hidden units: The number of hidden layers has significant impact on the performance. The RMSE on training and validation dataset with corresponding F (number of hidden units) is shown in table 1. Notice that these number is obtained with early stopping. It is obvious that RMSE drops as F increases. However, the gap between training RMSE and validation RMSE also increases. With larger gap between training and validation RMSE, the performance on testing dataset will be worse. We submit the result of 100 hidden units, the RMSE on testing dataset is around 1.07. In the second round, we submit the result of 50 hidden units, and the RMSE improves to 0.97.

Number of hidden units	25	50	100
Validation RMSE	0.79	0.63	0.53
Training RMSE	0.64	0.37	0.23

Table 1: performance of different number of hidden units

2) Analysis on learning rate: The value of learning rate could influence the performance as well. It could influence the value of RMSE and speed to converge. The influence of different learning rate is shown in table 2. The influence is mainly about convergence speed. The bigger the learning rate is, the faster it converges. The influence of learning rate on RMSE is much smaller than the influence on convergence speed.

Learning rate	0.01	0.05	0.1
Convergence time (epochs)	22	21	18
Validation RMSE	0.75	0.61	0.60
Training RMSE	0.57	0.34	0.33

Table 2: influence of learning rate

3) Analysis on momentum: momentum gives inertia to training, making it faster to converge. The value of the momentum cefficient (i.e. how much the current momentum relys on the past momentum) influences the convergece speed. We can see this in table 3. The convergence speed with 0.25 and 0.75 momentum coefficient is slower than the convergence speed of 0.5.

The reason is that, when the momentum coefficient is 0.75, the new momentum relies too much on the former momentum. when the momentum coefficient is 0.2, the new momentum relies too little on the former momentum. Both of the two cases lead to slower convergence.

Momentum Coefficient	0	0.25	0.5	0.75
Convergence time (epochs)	15	13	10	15

Table 3: influence of momentum

4) Anlysis on bias: adding bias makes model complete. Without bias, the RBM is not a complete model. Using bias helps improve the performance although the improvement is not very significant. The performance without biases is 0.68 and 0.41 for validation and training dataset when there are 50 hidden units. The performance with biases is 0.63 and 0.37 for validation and training dataset when there are 50 hidden units.

5) Analysis on early stopping and adaptive learning rate: early stopping and adaptive learning rate is used for dealing with verfitting. Overfiting occurs when the model keeps training. Figure 1 demonstrates the overfitting phenomenon. When the training continues, the training RMSE will usually keeps decreasing. However, the validation RMSE will not necessarily decrease all the time. It may first decreases, then fluctuates, and finally increases. To deal with this dilemma, we can first decrease the learning rate adaptively and then stop training.

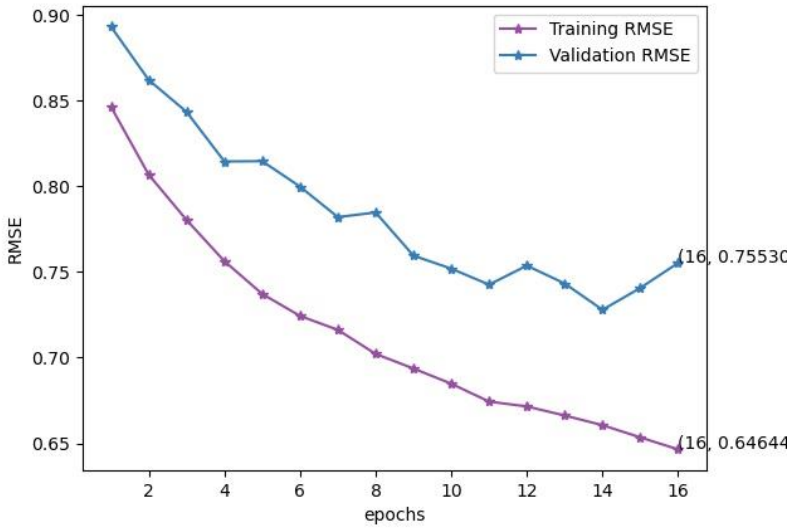


Figure 1: demonstration of overfitting

6) Analysis on regularization: as mentioned in theory part, regularization mitigates overfitting by punishing dense and complicated weight matrix. It does help to reduce overfitting. The influence of regularization is shown in table 4. The regularization coefficient cannot be too big or too small. If it is too big, the gradient descent will be dominated by regularization, the normal gradient descent will be diminished, which leads to bad performance on validation dataset. The regularization will surely lead to worse performance on training dataset. Since it can improve the validation performance, regularization is still desirable.

Regularization coefficient	0	0.01	0.1
Validation RMSE	0.79	0.75	0.89
Training RMSE	0.64	0.68	0.84

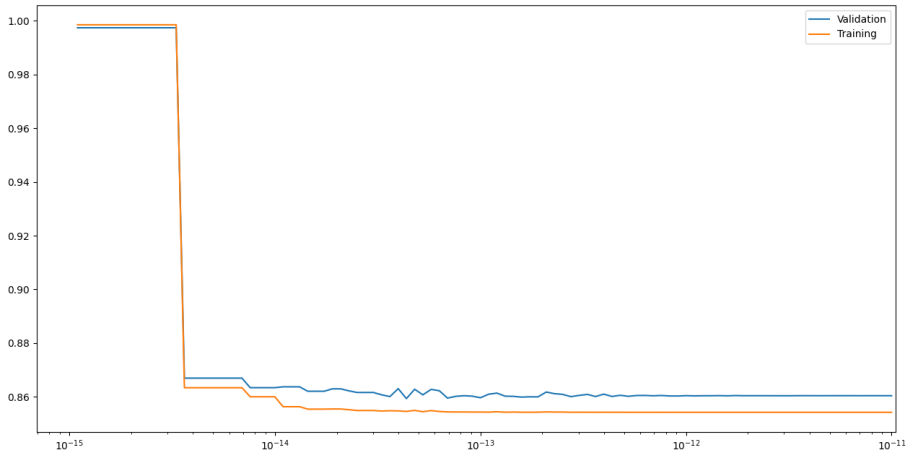
Table 4: influence of regularization

7) Analysis on mini-batch: the template does gradient descent for every user. Therefore, we may consider the original codes in template is mini batch size batch size 1. In this project, we have tried batch size of 1, 10 and 20. It is clear that the convergence speed decreases as mini-batch size increases. The reason is that smaller batch size will have higher gradient descent frequency. Hence, it will converge faster.

V. Discussion

A. Linear Regression

While the RMSE meets its minimum when λ=0.005, it might be restricted by resolution. Therefore, smaller λ are attempted to be implemented to find out when λ is too small to make contributions. 101 log uniform points are chosen between 10^{-15} and 10^{-11}. The results are as follows:



It could be noticed that critical change of RMSE occurred at about $\lambda = 3.31*10^{-15}$.

B. Restricted Boltzmann Machine

Besides all required RBM extensions, we also implement cross validation to further improve performance. The main idea is that we first shuffle training data and testing data, and then choose part of the total data as data used for left-out validation. Then, we will use the remaining data for cross validation. We divide the data into K parts. We also build up K estimators. The ith estimator will use the ith part of the data as validation set, and the rest as training set. In each epoch, we train on these K estimators with different validation set and training set. Then we will use K estimators to infer the result of left-out validation set (please notice that the left-out validation set is chosen before dividing total data into parts and it is not used in any estimator training). We will sum up the inferred results of all K estimators and get the average value as the final inference result.

Then, we can see the graph in figure 2. We can see that the left-out validation RMSE is smaller than the average validation RMSE in these K estimators. This makes sense since the average validation RMSE represents the average performance of one estimator, and the left-out validation RMSE represents the performance of the cooperation of K estimators. It is obvious that the performance of K estimators cooperation is better than one estimator.

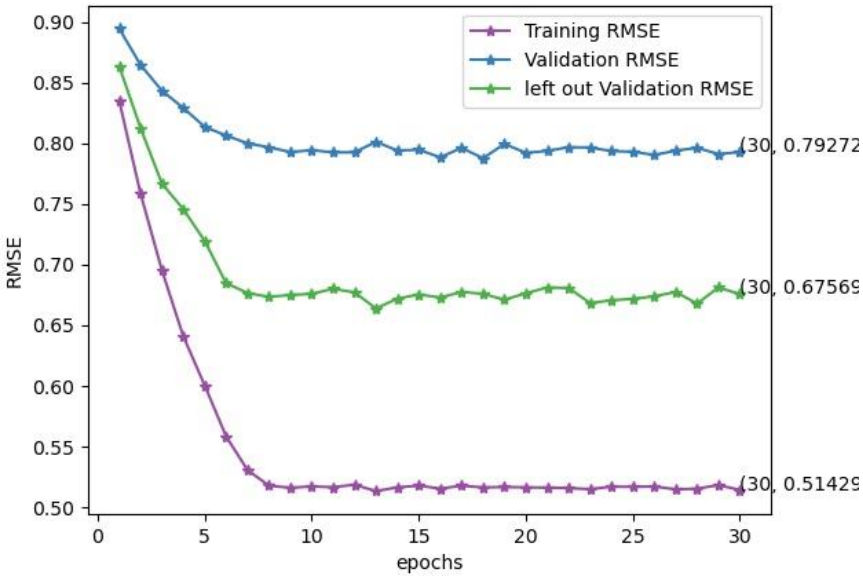


Figure 2: RMSE of cross validation

VI. Conclusion

For data with linear regularities, linear regression could be a convenient and efficient algorithm. However, the regularity in the Netflix dataset is far from only linearity. Therefore, more complex algorithms should be implemented for further accuracy.

Restricted Boltzmann Machine has better performance compared to linear regression because of the better structure of RBM. To get a better model, we need to solve the problem of overfitting. We used regularization, early stopping, adaptive learning rate, cross validation and other methods to solve overfitting problem.

With various methods, bug progress is made in this project. At first, the RMSE on testing dataset is 1.07. Then in second round, we improve it to 0.97. Then, with the help of cross validation, our best RMSE is 0.8717.