Movie Recommendation System based on Metadata and User Ratings

Yongshang Wu, Ziran Zhang and Jiaxi Chen
Department of Computer Science
Stanford University
{wuy, zirzhang, jiaxi}@stanford.edu

Abstract

Movie recommendation is a challenging but interesting problem, where data sparsity and *cold start* issues make prediction very hard. We proposed an AutoEncoder based deep learning model as well as a embedding based score model to tackle the movie recommendation problem given user ratings and movie metadata. Our AutoEncoder model reaches 0.496 on r^2 metric and 74.3% prediction accuracy on the test set.

1 Introduction

Movie recommendation is a interesting problem given the large amount of movies available and users in need. A recommendation system is required to discover users' latent preference over movies by analyzing the existing (often a small fraction) ratings of users over movies. In this project, we hope to build an end-to-end recommendation system that could predicts users potential ratings on movies given given existing movie ratings and movie metadata.

2 Related Work

Collaborative Filtering (CF) [1] [2] is a popular and powerful technique employed by most recommendation systems. Traditional methods of CF like matrix factorization [3] with principal component analysis (PCA) have been widely employed to tackle this problem. Neural network based approach such as Neural Collaborative Filtering [4] is gaining more and more popularity with the emerging and rising of deep learning.

Network embedding, such as node2vec [5], is another well-known method to represent relationship of nodes in graphs.

But these work either focuses only on rating matrix or movie/user metadata, and it is often difficult to combine the advantages of two methods and hard to be integrated into applications. In addition, these techniques suffer from sparsity problem, also known as the *cold start* issue. It also takes lots of efforts or even is impossible to incorporate any domain knowledge or side information.

In this project, we designed a AutoEncoder [6] based method which combines ratings and movie metadata to tackle the recommendation task.

3 Dataset

The dataset we uses is obtained from 'The Movie Dataset [7]. After preprocessing, we have 2,095 movies and 7,941 users with around 1 million ratings and each movie comes with metadata including overviews, keywords and genres. So the input we end up with is a (2095, 7941)-shaped rating matrix

R and a (2095, 300)-shaped metadata matrix M obtained by averaging word vectors of textual metadata.

4 Model

4.1 Denosing AutoEncoder Model

Figure 1 is the illustration of processing steps in the Denoising AutoEncoder (DAE) model.

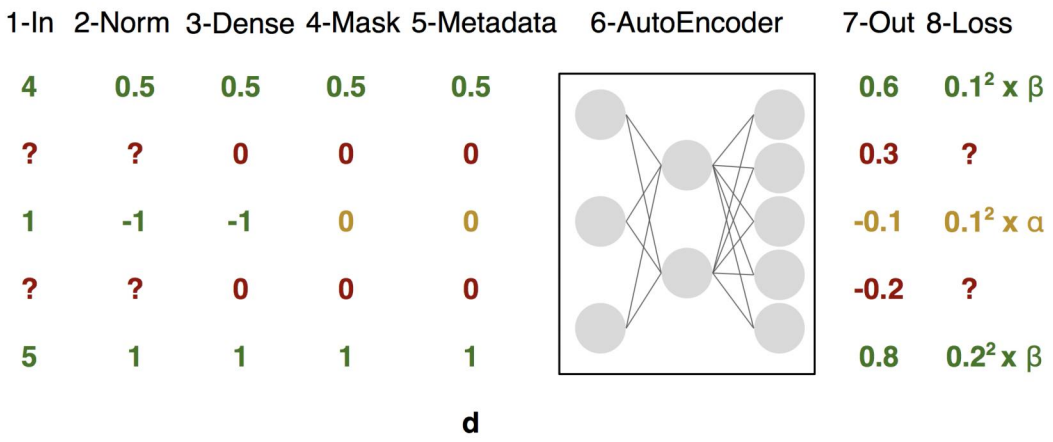

Figure 1: Denoising AutoEncoder model

First, given the input movie vector, we normalize the ratings to a $[-1, 1]$ scale, which allows a tanh activation output layer to output ratings at the same scale without additional post-processing.

Second, we densify the vector by filling unknown ratings with 0's and the index of these missing values needs to be recorded to inhibit the back-propagated unknown errors. No error should be back-propagated for missing values since we don't know the true "label" of them, while the error on actual zero values are back-propagated normally.

Third, 30% of random known values are masked to 0 to enable the usage of Denoising AutoEncoder loss function [8].

Next, the d-dimensional metadata m_j of the j-th movie is appended to the vector. The consideration for incorporating metadata of movies is that it may help alleviate "the cold start" problems when there are very few users rating on particular movies. Adding the intrinsic side information of movies may add some *bias* on the missing ratings as a basis start point. In our neural network implementation, such method could be applied by appending metadata's vector representations to the original input vectors. For baseline model, we apply the CBOW [9] technique by averaging up word vectors of movie overviews, keywords and genres as the metadata and only apply the metadata at the input level. Different metadata representations and applications at different layers could be also tried out and compared.

In terms of the AutoEncoder, We first implement a shallow neural network with one hidden layer to get a baseline model. The predicting task is achieved by performing a forward pass through our neural network:

$$\hat{R}_{\cdot j} = g^{[2]}\left(W^{[2]} \cdot g^{[1]}\left(W^{[1]} \cdot [R_{\cdot j}, m_j] + b^{[1]}\right) + b^{[2]}\right) \tag{1}$$

where $g^{[1]}, g^{[2]}$ are non-linear activation function for different layers, $W^{[1]} \in \mathbb{R}^{n^{[1]} \times N}, b^{[1]} \in \mathbb{R}^{n^{[1]}}, W^{[2]} \in \mathbb{R}^{M \times n^{[1]}}, b^{[2]} \in \mathbb{R}^{M}$ are the weights and biases to be learnt in the network. For baseline, we choose the activation function $g^{[1]} = ReLu, g^{[2]} = tanh$ and the hidden layer size $n^{[1]} = 750$. More architecture details and choices carefully calibrated in section 5.

As mentioned before, we define the DAE loss function as

$$\mathcal{L}_{\text{DAE}}(R_{\cdot j}, \hat{R}_{\cdot j}) = \alpha \sum_{i \in \mathcal{E}(R_{\cdot j}) \cap \mathcal{M}(R_{\cdot j})} \left(\hat{R}_{ij} - R_{ij}\right)^2 + \beta \sum_{i \in \mathcal{E}(R_{\cdot j}) \setminus \mathcal{M}(R_{\cdot j})} \left(\hat{R}_{ij} - R_{ij}\right)^2 + \lambda \|\mathbf{W}\|_2^2 \tag{2}$$

where $\mathcal{E}(R_{\cdot j})$ is the set of indices of existing values in $R_{\cdot j}$ and $\mathcal{M}(R_{\cdot j})$ is the set of indices of masked values. The intuition behind the DAE loss function is that we want to encourage our model to not only **recovering** the known values, but also **predicting** unknown values. Consider a model which learns to simply output the original input vector (i.e. an identity function), the loss of this model is 0 but is useless since it can not conduct any predicting. Without masking, it is highly likely an AutoEncoder would learns the behavior of the example identity model.

4.2 Score Model

Inspired by word2vec [9], we came up with the Score Model, which learns denser representation embeddings for movies and users, as illustrated in Figure 2.

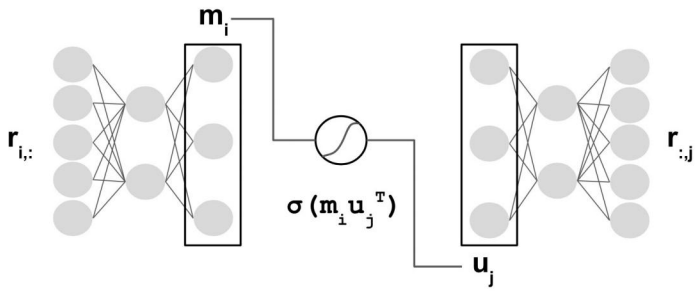

Figure 2: ScoreModel with Neural Network

The Score Model is the focus of Ziran and Jiaxi's joint project for CS224N, which will not be discussed in detail here due to the page limit.

5 Results and Analysis

5.1 Metric

5.1.1 Coefficient of Determination

We use coefficient of determination (r^2) [10] as the metric of evaluating our AutoEncoder model instead of Root Mean Square Error (RMSE) used in our milestone. They are both metrics for regression problems but it is hard to interpret the RMSE metric since it is just an unnormalized error. The r^2 metric is defined as follows:

$$r^2 = 1 - \frac{\sum |y - \hat{y}|^2}{\sum |y - \overline{y}|^2} \tag{3}$$

which lies in range $[-1, 1]$ and indicates the percentage of improvement over a baseline which simply output mean value of input values, as illustrated in Figure 3.

5.1.2 Prediction Accuracy

The movie recommendation task can be treated a binary classification problem: essentially we want to do is predict whether to recommend a movie to a user ($y = 1$) or not ($y = 0$). This could be achieved by applying a threshold (e.g. rating ≥ 4) to the ground truth and predicted ratings. So a natural metric of accuracy ACC could be adopted under the binary classification setting:

$$ACC = \frac{\text{\# of Corrected Predictions}}{\text{\# of Predictions}} \tag{4}$$

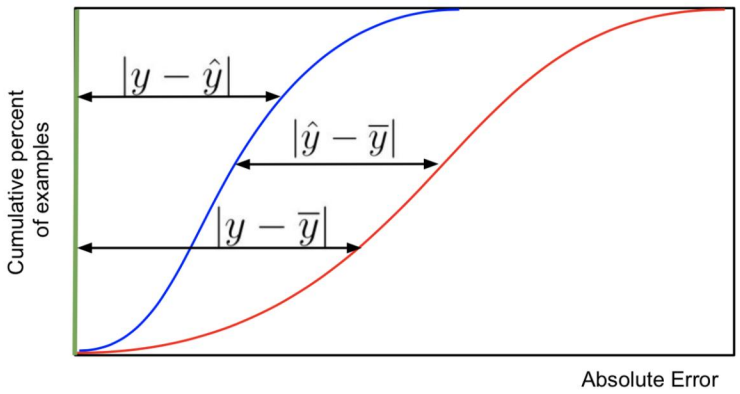

Figure 3: Coefficient of Determination Metric

5.2 Experiment Resutls

We randomly split the dataset into $80\% : 20\%$ training/dev set to do training, cross validation and hyperparameter tuning. In all experiments, we run our models for 100 epochs, using AdamOptimizer [11] with a learning rate of 10^{-3}, a l_2-regularization weight of 0.1, and a dropout rate of 0.8. For details of all hyperparameters and our code implementation, please refer to the Github Repo[1].

5.2.1 DAE Model Results

Figure 4 shows the results we get for different autoencoder architectures. We start with one 750-

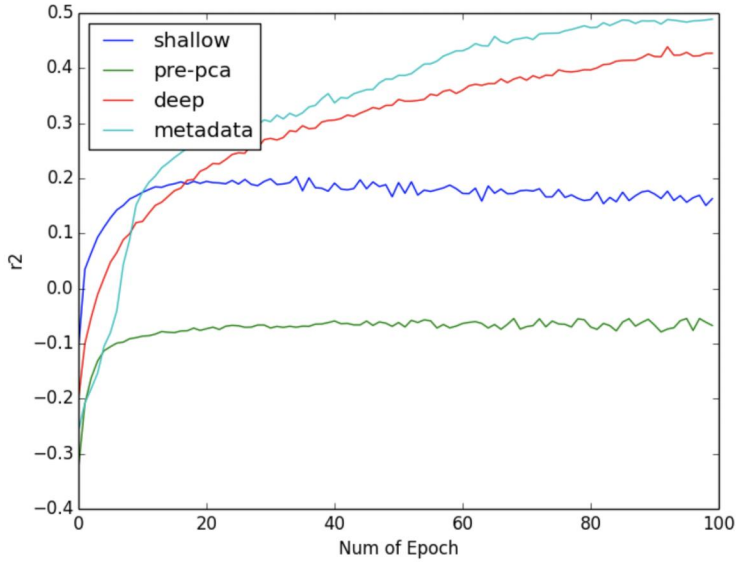

Figure 4: Experiment Results of DAE Model

neuron hidden layer without metadata. As we can see, the model overfits quickly and the r^2 metric on dev set gets stuck around 0.167 (the "shallow" line in Figure 4). The reason may be that we have a very high dimensional input and output, which results in a huge amount of parameters and easily overfits the training data.

To address the overfitting problem, we first try to reduce the dimension of input vector by doing a principal component analysis on the input matrix, then apply the autoencoder to the compressed

[1] `https://github.com/AndyYSWoo/cs230-MovieRecSys`

vectors (the principal components) and then recover the output back to the original dimension space. But this approach makes the performance even worse, with a r^2 metric around -0.1 (the "pre-pca" line)! The possible reasons are: 1) the model bias is enlarged by trying to recover the imprecise compressed components; 2) the principal components for train/dev sets are different so using the same transformation is probably a bad idea.

We then try decreasing the first hidden layer size from 750 to 50 and stack a second hidden layer of size 100, followed by another hidden layer of size 50. This boosts the dev r^2 metric to 0.418 (the "deep" line). The intuition behind this architecture is that by decreasing the size of first and last hidden layer we could decrease the number of total parameters; by stacking three hidden layers instead of having a single large layer, the model will be able to learn denser features of the movie vector.

Lastly, by concatenating movie metadata to the input vector, the r^2 is increased to 0.496.

5.2.2 Score Model

By trying out different combinations, we achieve the best result on the Score Model with 50-dimensional embedding vector, 2 hidden layers, and 60 neurons in each hidden layer. The experiment results is as shown in Table 1:

Table 1: Accuracy on different models

Model	Accuracy
Score Model without neural network	62.9%
Score Model with neural network	70.9%
Score Model with neural network plus metadata	71.1%
DAE Model	70.2%
DAE Model plus metadata	74.3%

We observe that even with a relatively shallow neural network with 2 hidden layers, the Score Model still outperforms the one without hidden layers and the metadata has less effect on the performance than DAE Model.

Due to the page limit, we will not discuss the results of Score Model here since it is the focus of the CS224N joint project.

6 Conclusion and Future Work

In this project, we came up with and implemented two end-to-end deep learning based recommendation system models, namely the Denoising AutoEncoder Model and the Score Model. We devoted large amount of efforts to tackle the overfitting problem of the DAE model and learned a lot from the process.
Our future work can be looking into the combination of traditional recommendation methods and our model into some hierarchy statistical model. Another aspect of future work lies in different ways of utilizing the textual metadata with recurrent neural networks and maybe image metadata (e.g. posters or movie content frames) with convolutional networks.

Contribution
Yongshang Wu: Denosing AutoEncoder Model design and implementation, experiment management, score model discussion, poster making and report writing.
Jiaxi Chen: Score Model design and implementation, experiments on Score Model, data preprocessing, poster making and report writing.
Ziran Zhang: Research on related works, investigation on state of the art, discussion of dataset and algorithm, poster making and report writing.

Shared project
Jiaxi and Ziran share this project with CS224N, where they get the inspiration of the Score Model from and pay most attention to.

References

[1] Will Hill, Larry Stead, Mark Rosenstein, and George Furnas. Recommending and evaluating choices in a virtual community of use. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 194–201. ACM Press/Addison-Wesley Publishing Co., 1995.

[2] Joseph A Konstan, Bradley N Miller, David Maltz, Jonathan L Herlocker, Lee R Gordon, and John Riedl. Grouplens: applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3):77–87, 1997.

[3] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8), 2009.

[4] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, pages 173–182. International World Wide Web Conferences Steering Committee, 2017.

[5] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[6] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 37–49, 2012.

[7] Martin E Fabien D, Rounak B. The movie dataset. `https://www.kaggle.com/rounakbanik/the-movies-dataset`, 2017.

[8] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[9] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[10] Calyampudi Radhakrishna Rao, Calyampudi Radhakrishna Rao, Mathematischer Statistiker, Calyampudi Radhakrishna Rao, and Calyampudi Radhakrishna Rao. *Linear statistical inference and its applications*, volume 2. Wiley New York, 1973.

[11] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.