Probabilistic Models and Machine Learning: HW 3 Report

Yusuf Efe (me2799)

November 18, 2022

Problem 1) In this homework, I worked on Matrix Factorization to work on a movie recommendation problem. Firstly, I chose the MovieLens data set to work on. In the particular set I used, there are totally 610 distinct users and 9724 distinct movies. However, not everybody watched and rated each movie in the data set. The statistics about the data set are given below:

$$\begin{cases} \text{Number of movies: } 9,724 \\ \text{Number of users: } 610 \\ \text{Number of rating: } 100,836 \\ \text{Movies x users: } 5,931,640 \\ \text{Rated movie ratio: } \approx 1.7\% \text{Average of rated movie count by each user: } 165 \\ \text{Std of rated movie count by users: } 269 \end{cases}$$

The underlying principle for using matrix factorization algorithm is that we will try to create equal length feature vectors for each user and movie. These latents will try to approximate the number of features that represent the movies and users domain in the best possible way. For example, these latents may mean the genre of the movie: such as drama, action, romantic, horror, etc. Also, they can mean different features such as the language, length, and effects of the movie. In the movie side, these vector values will mean the strength of the features present in the movie. As it will be shown later, the algorithm is designed in a way that all the features are almost zero mean across different movies. As for the user side, the features mean the degree of the importance the users assign to the corresponding features. Being zero means "do not mind" from the user's perspective for the corresponding movie feature. While positive values mean that the user favors this feature, negative values mean that the user dislikes it.

Training Procedure:

During the training, I used the "pypi" Matrix-Factorization library developed by "Quang-Vinh Do". Firstly, I split the 0.2 percent of the data for testing our model's rating approximation. For the training part, I firstly pretrained the model by using a small portion of the all training set (already rated movies by user), to help the vectors converge to a local minimum corresponding to the initial set. After that, I used the all training set to complete the training schedule. For the training, rather than doing applying gradient descent directly, I used the stochastic gradient descent algorithm to give model a chance to escape from the local optimum points.

First, using the vector size as 100, and training the model for 20 epochs, training RMSE graphs w.r.t. epoch is found as follows:

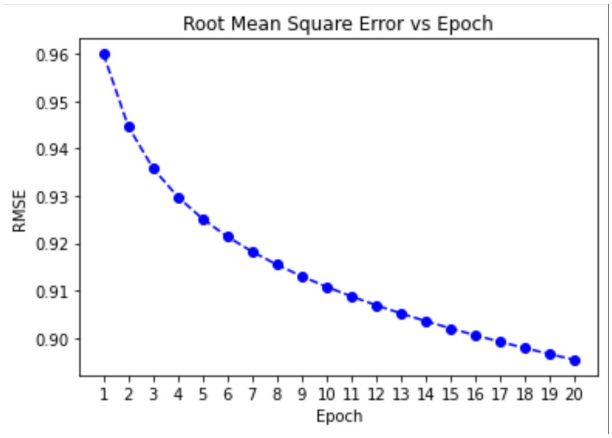

Figure 1: RMSE of Training Data - Epoch-20 and Vector-Size-100

Resulting Accuracy:
For this setting: while the resulting RMSE for the training data is: 0.895, the resulting RMSE for the test set is: 0.915. Thus, it is good to see that the model dod not overfit and the discrepancy between the results is low.

Here, we also investigated the resulting latent vectors and finalized gradings. Since it is not viable to illustrate a 100-dimensional vector, we will show the average cos distance between the users and the 5-rated movies. For the opposite side, we will show the average cos distance between the users and the 0.5-rated movies (lowest). Here are these two key statistics in the resulting table:

$$\begin{cases} \text{the average cos distance for the 5-rated movies: } \mathbf{0.24} \\ \text{the average cos distance for the 0.5-rated movies: } \mathbf{-0.37} \end{cases}$$

Here, we can see that the cosine distance between the users and their probable favourite movies are low (similar vectors), but the cosine distance between the bad matched user-movie vectors are generally opposite oriented w.r.t. each other, thus having negative values. Some other statistics about the model is that, the average grading for both already-rated movies and grading-predicted movies are 3.5, which shows that the model uses this 1.7% percent of this grading data as a good representation of the all possible matchings and carry the statistics to them, too. Also, as can be seen in the code shared, the mean values for each feature entry among users and movies are very close to zero.

After this, the last examination we could do is changing the length of feature vectors to see the effect of it on model performance/underfitting-overfitting relations.

Here, we will run the model for 30 epochs and check its performance for different K-vector sizes: $[5, 10, 20, 50, 100, 200]$. The resulting training processes and reached test-accuracy values are presented below:

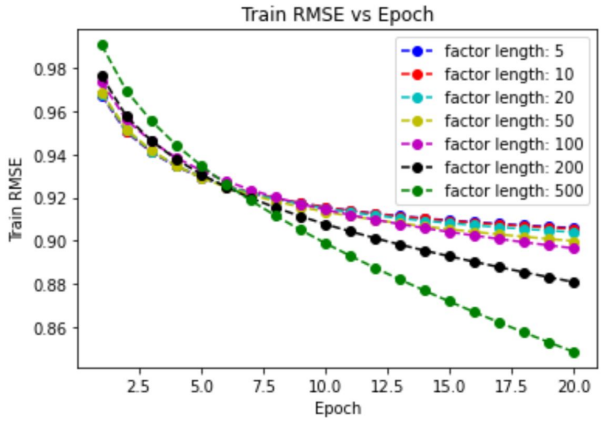

Figure 2: Training RMSE Values for different K-vector sizes

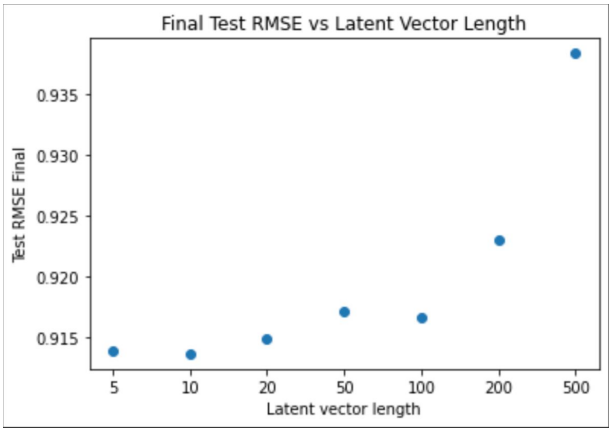

Figure 3: Final test RMSE values after 20 epoch for different K-vector sizes

Here, we can see that the training accuracy gets better as the number of features increases. It simply shows the overfitting when we also check results for the final RMSE errors on test set. That is why, it is better to choose K = 10 for the best generalization.