CSE2525 DATA-MINING CHALLENGE REPORT

By Rami Al-Obaidi

Abstract

Evaluation of multiple Collaborative Filtering algorithms for movies recommendation system

Date: 15 February 2021

Table of Contents

Introduction

The goal of this challenge is to make a recommender system to predict ratings for movies in Python.

The data used for this challenge consists of the following:

- **6,040 users** defined with their:
  - **gender**
  - **age**
  - numerical representation of their **profession**
- **3,706 movies** defined with their:
  - **title**
  - **year of release**
- **910,190 ratings** on a **1 to 5 scale** given by the users to movies

There are two main Collaborative filtering algorithms which I have used for this challenge:

1. **Nearest-Neighbours (Item-based)**
2. **Latent Factor Models**

The focus of the report is to make comparisons between these two algorithms in terms of their **accuracy**. The data set used for this challenge is not small and these two algorithms will use a lot of time to make the predictions.

To evaluate our recommender system, there are **90,019** withheld ratings with which our recommender system needs to make predictions. These predictions are then compared to actual ratings and an error value is calculated to represent the accuracy of the prediction algorithm. The error metric used for this challenge is **Root-Mean-Square Error** (RMSE) which is calculated as follow:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(P_i - O_i)^2}{n}}$$

$P =$ predicted value

$O =$ actual value

$n =$ number of values

Equation 1: Root-Mean-Square Error

Nearest Neighbour (Item-based)

Methodology

The algorithm used for the challenge is defined as follow for a rating of movie **i** by user **x**:

- For movie **i**, calculate the similarities to other movies that have been rated by user **x**
- Pick the highest **k** movies based on the calculated similarities (neighbours)
- Predict using the score function:

$$\widehat{r_{xi}} = b_{xi} + \frac{\sum_{j \in N(i;x)} s_{ij} \cdot (r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} s_{ij}}$$

$\widehat{r_{xi}} =$ rating of movie j by user x
$s_{ij} =$ similarity between movies i and j
$b_{xi} =$ baseline estimate for r_{xi}

Equation 2: score function

The **baseline estimate** is used to model the global and local effects and is calculated as follow:

$$b_{xi} = \mu + b_x + b_j$$

$\mu =$ overall mean rating

$b_x =$ (average rating of user x) $- \mu$

$b_i =$ (average rating of movie i) $- \mu$

Equation 3: baseline estimate

The **similarity metric** used in this algorithm is Pearson's Correlation and is defined as:

$$sim(X, Y) = \frac{cov(X, Y)}{\sigma_X \sigma_Y}$$

Equation 4: Pearson Correlation

Advantages

It is easy to implement and there is no training phase.

Disadvantages

It suffers from cold star (no enough users), first-ratter (can't recommend an unrated item) and tends to be biased towards popular items.

Result

Here is a graph that compares the RMSE of this algorithm with 3, 5, 7, 9 and 12 nearest neighbours.

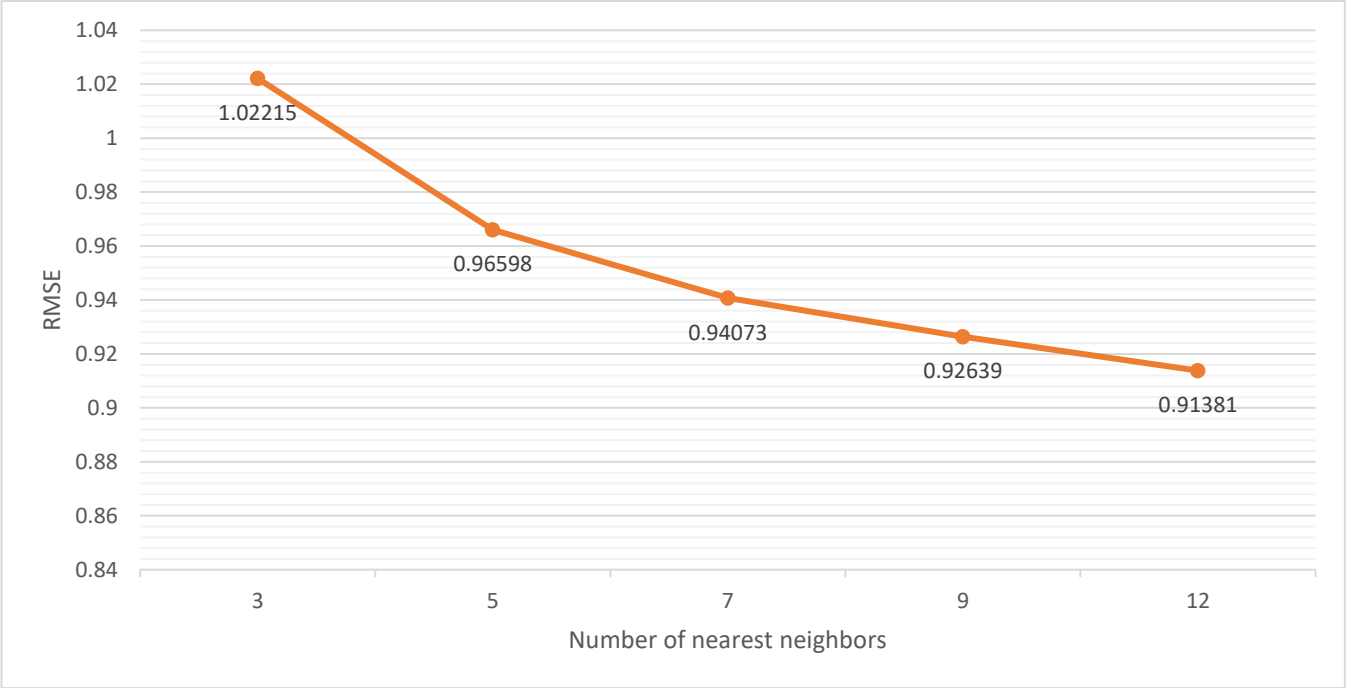

Figure 1: Nearest neighbours RMSE

The RMSE decreases with an increase in number of neighbours, However, choosing a high number of neighbours will lead to overfitting and the number of neighbours should be chosen with caution.

Performance Evaluation

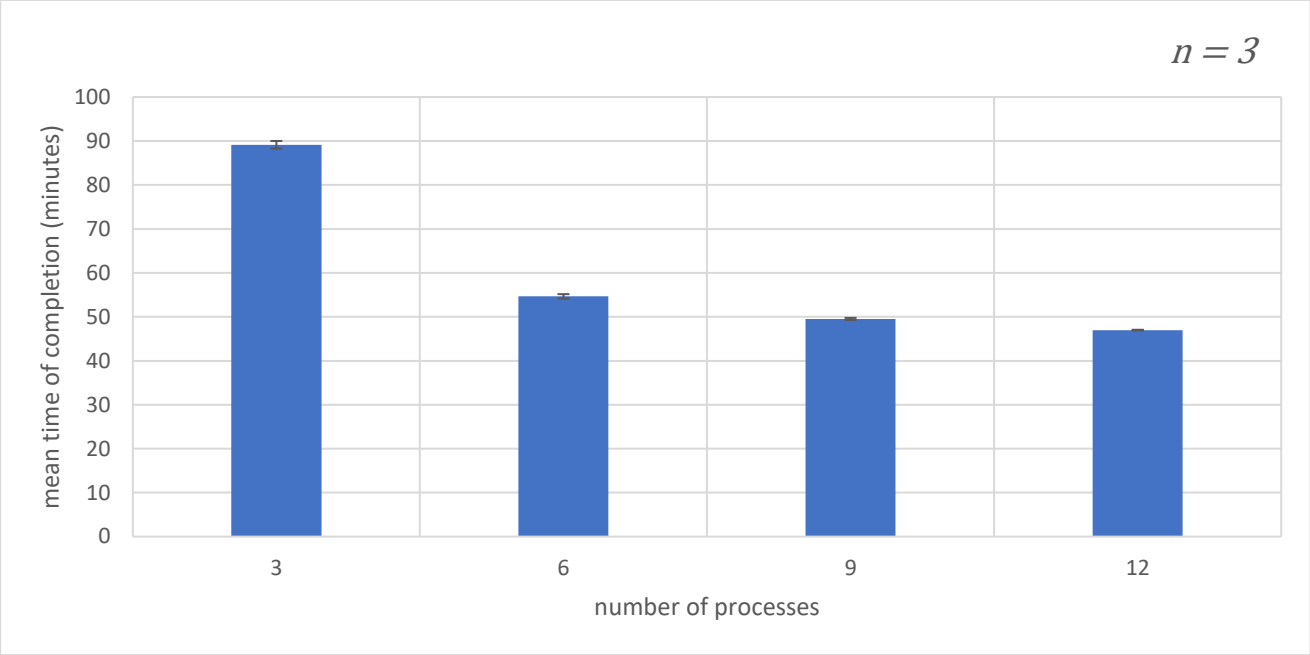

Figure 2: time of completion for nearest neighbour

This algorithm benefits greatly from multiprocessing. The time of completion decreases by **40%** when going from 3 processes to 6 processes. However, the **limiting factor** here is the CPU core count (6 cores) so there isn't a high decrease in time after 6 processes. The standard deviation (shown as error bars in the graph) is very low which is a good indicator for the quality of the experiment.

Latent Factor Models

Methodology

It is possible to decompose our original utility matrix into two matrices Q (number of items x number of factors) and P^T (number of factors x number of users). These factors represent different categories, for example, comedy or action for movies and males or females for users.

We can use **Singular Value Decomposition** (SVD) to construct these matrices. These matrices need to have the minimum reconstruction error (Sum of Squared Errors) which SVD is known to have. However, SVD is not defined when there are undefined entries.

Instead, we use stochastic gradient descent to construct Q and P^T while minimizing the reconstruction error. The matrices are first initialized with random entries then we use stochastic gradient descent to find the optimal matrices.

However, this can lead to *overfitting*. To mitigate this, we introduce regularization λ.

Like the Nearest-Neighbour algorithm, we will also take into account the global and local effects while performing Stochastic Gradient Descent.

Now, Stochastic Gradient Descent will used for estimating $b_x,\ b_i,\ q_i$ and p_x by:

$$\min_{Q,P} \sum_{(x,i)\in R} (r_{xi} - (\mu + b_x + b_i + q_i \cdot p_x))^2 + (\lambda_1 \sum_i \|q_i\|^2 + \lambda_2 \sum_i \|p_x\|^2 + \lambda_3 \sum_i \|b_x\|^2 + \lambda_4 \sum_i \|b_i\|^2)$$

Equation 5: Sum of Square Errors for the model

We will use the same regularization factor for all the parameters to simplify the process and reduce the number of parameters we need to set in our evaluation.

Here is the pseudo code of the learning stage that I implemented:

$Intialize\ Q\ and\ P^T\ with\ random\ entries$

$\alpha\ := the\ learning\ rate\ of\ stochastic\ gradient\ descent$

$\lambda := regularization$

for each $iterations\ of\ stochastic\ gradient\ descent:$

for each $know\ rating\ \ r_{xi}:$

$\widehat{r_{xi}} := \mu + \ b_x + b_i + q_i \cdot p_x$

$error\ :=\ \widehat{r_{xi}} - r_{xi}$

$b_i :=\ b_i + (\alpha\ \cdot (error - (\lambda \cdot b_i))$

$b_x :=\ b_x + (\alpha\ \cdot (error - (\lambda \cdot b_x))$

$q_i :=\ q_i + (\alpha\ \cdot ((error \cdot p_x) - (\lambda \cdot q_i)))$

$p_x :=\ p_x + (\alpha\ \cdot ((error \cdot q_i) - (\lambda \cdot p_x)))$

end for

end for

Figure 3: Pseudo code for Latent Factor Models

Then the rating for user **x** and movie **i** will be calculated as follow:

$$r_{xi} = \mu + \ b_x + b_i + q_i \cdot p_x$$

Equation 6: score function

Advantages

Quicker and more accurate than the nearest neighbour algorithm.

Disadvantages

Harder to implement. More parameters to deal with.

Results

number of factors	learning rate	regularization	learning iterations	RMSE
50	0,01	0,02	35	0,91326
30	0,01	0,02	20	0,90245
40	0,01	0,02	25	0,90594
50	0,01	0,02	25	0,92513
75	0,001	0,02	30	1,00455
15	0,001	0,02	30	0,92217
15	0,0075	0,02	30	0,87390
15	0,01	0,02	35	0,87816
15	0,005	0,05	45	0,86673
15	0,0025	0,05	60	0,87128
12	0,0075	0,075	30	0,87108
17	0,005	0,05	35	0,86766
10	0,005	0,05	35	0,86980
12	0,005	0,05	25	0,87722
12	0,01	0,05	20	0,86897
12	0,015	0,05	20	0,86778
12	0,02	0,08	20	0,87170
12	0,02	0,025	20	0,88140
12	0,005	0,05	45	0,86486
12	0,001	0,05	60	0,89931
12	0,005	0,05	60	0,86157
12	0,005	0,05	120	0,85918
12	0,005	0,05	240	0,85780
12	0,0075	0,05	360	0,85995
12	0,005	0,05	500	0,85628
12	0,005	0,05	1000	0,85719

Table 1: RMSE measurement for Latent Factor Models

From the table, we can see that the number of latent factors for the data set is between 10 and 15 factors. The RMSE becomes way higher when the number of latent factors is outside the range due to the data set consisting of a similar number of categories.

Control Experiments

In order to see how each parameter affects the RMSE independently, I conducted control experiments where one parameter is modified and the other three remain constant. The four graphs show how the number of iterations, learning rate, regularization and number of factors affect the RMSE independently.

This is the graph for the number of iteration of stochastic gradient descent. The number of latent factors was set to 12, the learning rate is 0.005 and the regularization is 0.05.

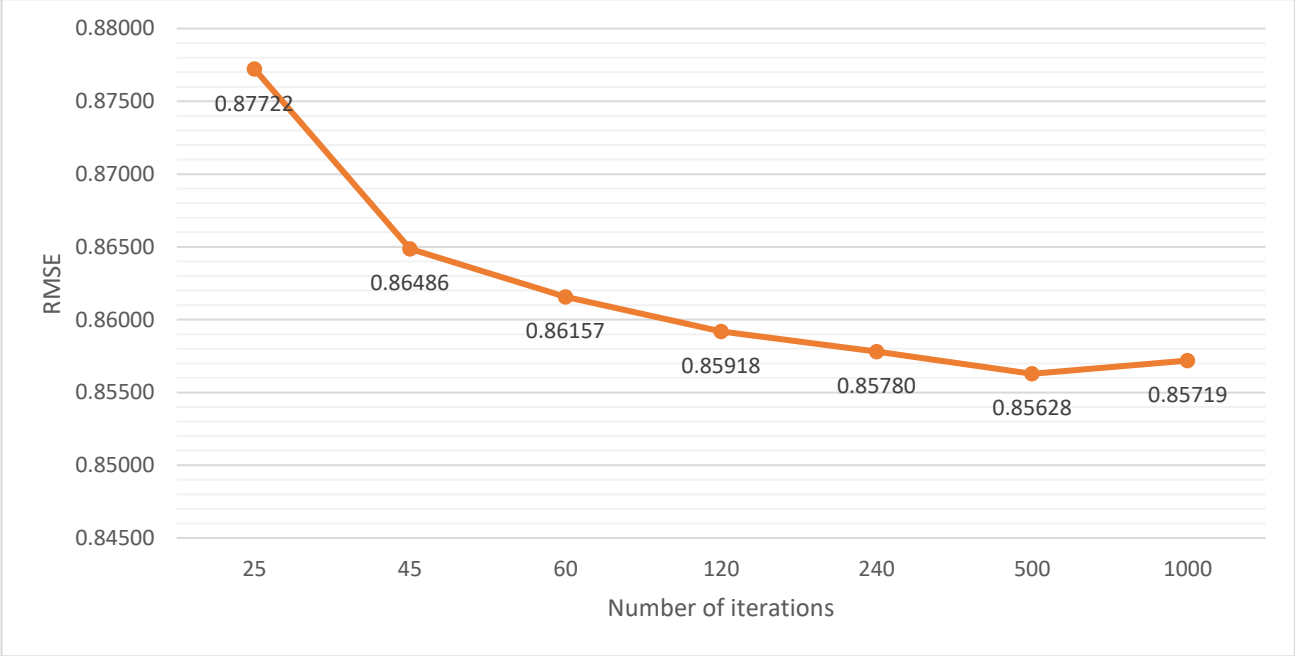

Figure 4: Effect of the number of iterations of stochastic gradient descent

Here is the graph showing how the learning rate of stochastic gradient descent affects the RMSE. The number of factors is 12, the regularization is 0.05 and the number of iterations is 25.

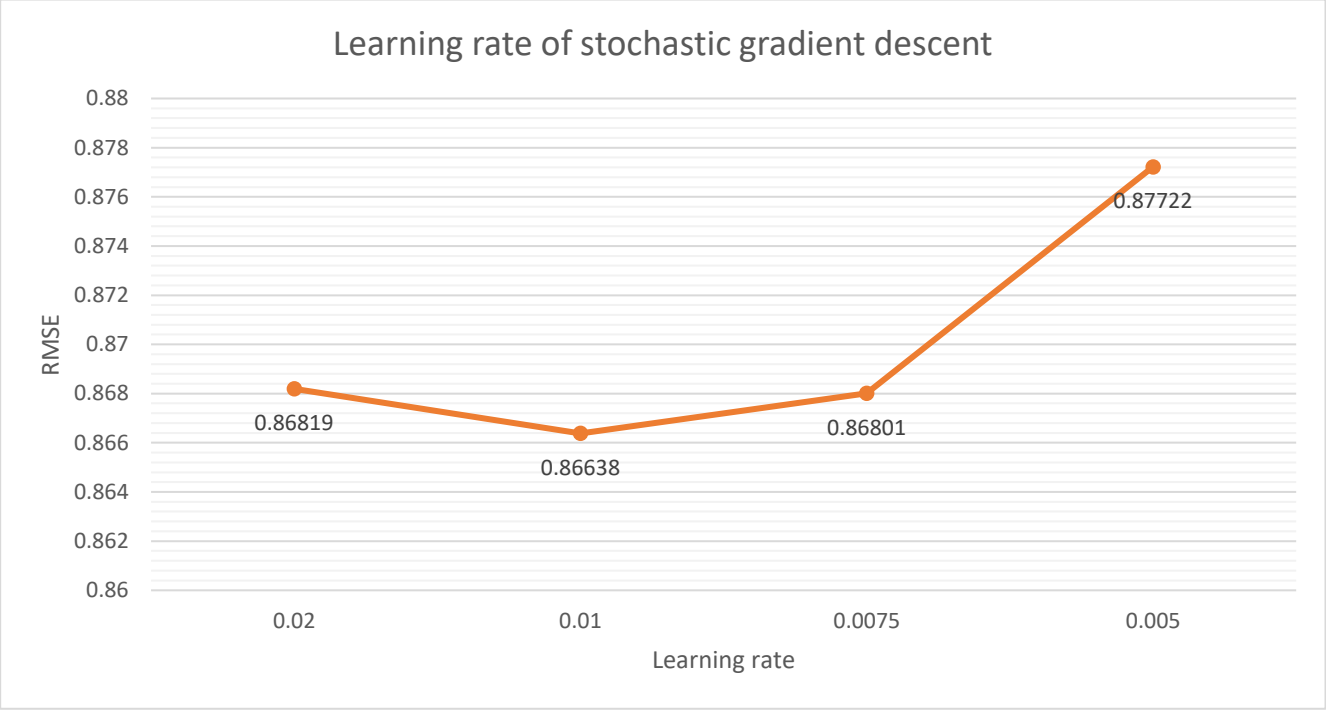

Figure 5: Effect of learning rate of stochastic gradient descent

For the regularization, the number of latent factors was set to 12, the learning rate to 0.02 and the number of iterations to 25.

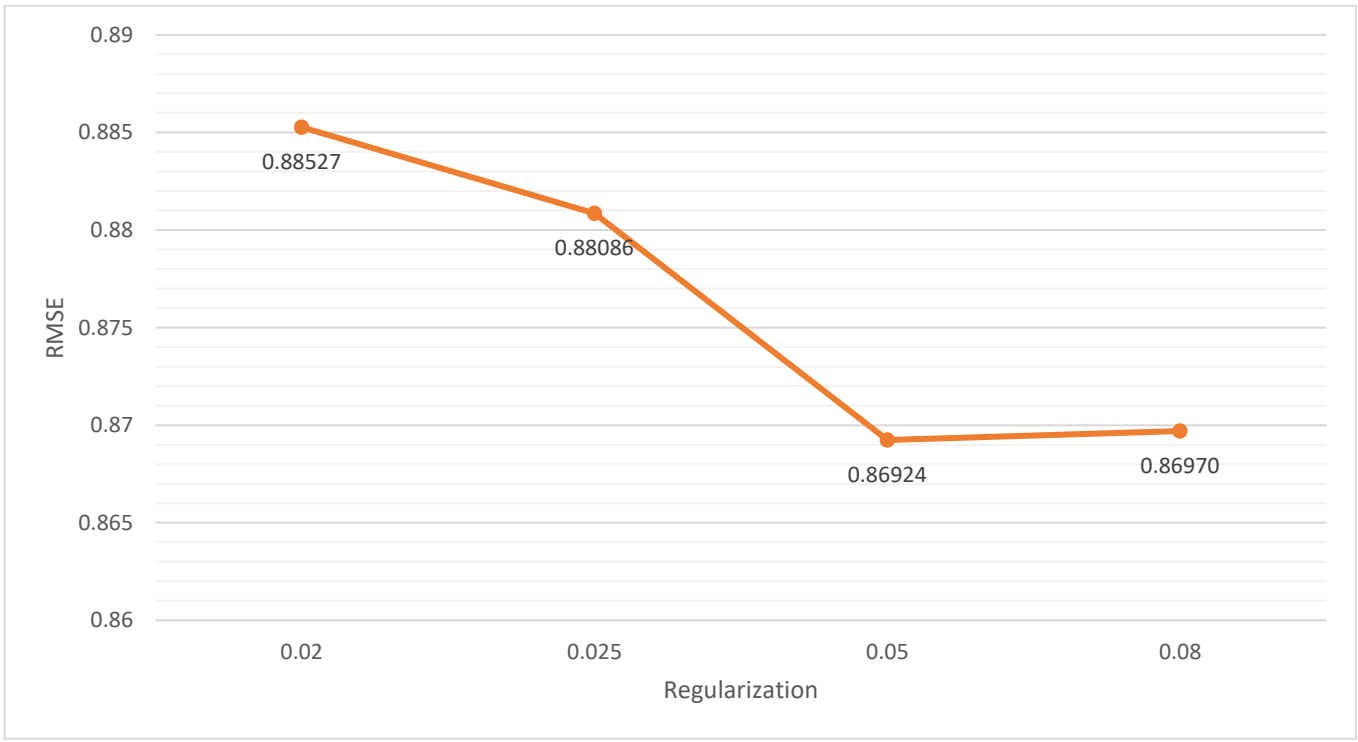

Figure 6: effect of regularization

For the number of latent factors, the learning rate is 0.005, the number of learning iterations is 25 and the regularization is 0.05.

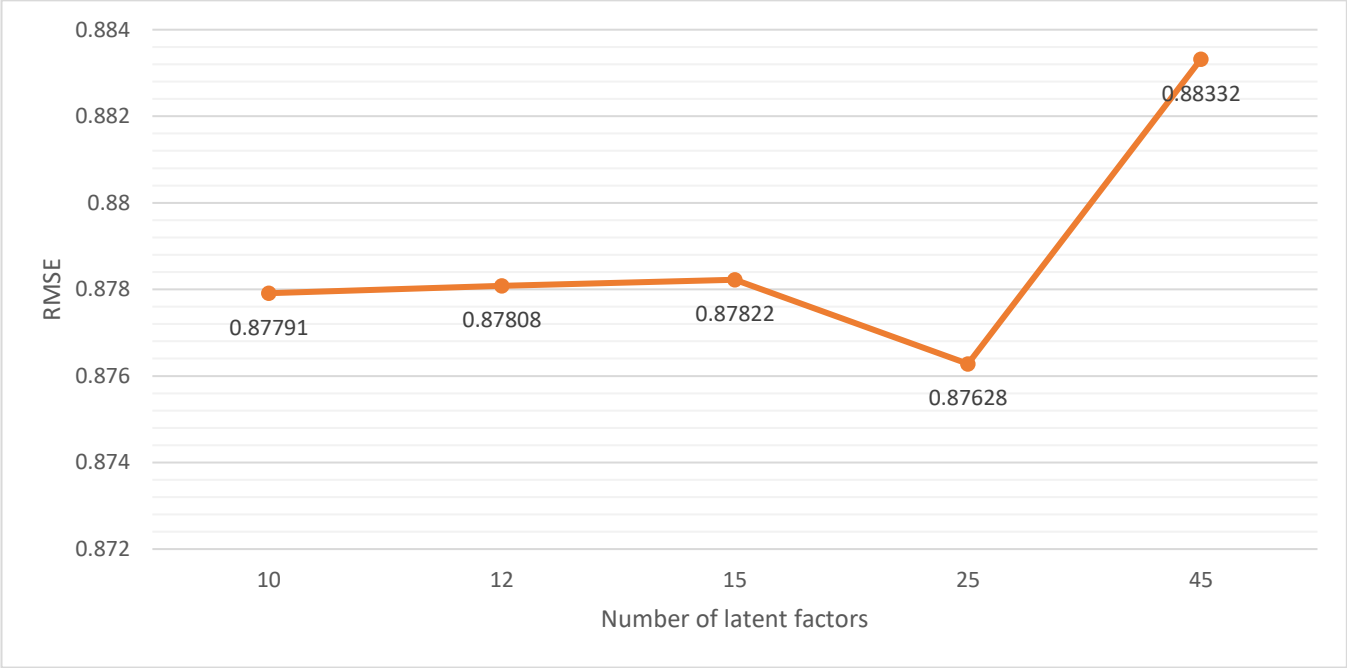

Figure 7: effect of varying the number of latent factors

Conclusion

The Latent Factor Models is a clear winner over nearest the neighbours algorithm not just in accuracy but also in performance. However, it takes a lot of effort to optimize the settings for Latent Factor Models.

Latent Factor Models consist of multiple settings (number of iterations, learning rate, regularization, number of latent factors) which affect the accuracy.

Multiprocessing can greatly improve the performance of the nearest neighbours algorithm as it is very inefficient.

Future Improvements

Improvements to Latent Factor Models are to use separate regularization for each hyper parameters and to use cross-validation to get better estimates on how much the regularization should be.

Adding temporal effects to Latent Factor Models which show the variation of ratings over time will improve the accuracy of the algorithm. However, our data set lacks the time of the ratings.

Different models can also be combined to better predict rating.