COMS 4771: Machine Learning (Fall 2018) - Homework #2

Amal Alabdulkarim (aa4235), Danyang He (dh2914), Jing Qian (jq2282)

November 3, 2018

Problem 1

i)

Full Gradient Descent[**bgd**, **Ruder**] only updates the parameters after it has evaluated the full training dataset. It computes the gradient of the cost function of the each example in the training dataset and updates the parameters with respect to θ, using the following equation:

$$\theta_t = \theta_{t-1} - \eta \nabla L(\theta) \tag{1}$$

The advantage of full GD is:

- It guarantees that this algorithm will find global minima for convex loss functions and local minima for non-convex functions
- Fewer updates are performed, which means this algorithm is more computationally efficient when compared to Stochastic Gradient Descent.
- More stable error gradient due to the low frequency of updates.

The weaknesses of full GD are:

- The algorithm updates are very slow due to the calculation of the gradients of the full dataset in every update which affects the training speed. Plus, because large datasets usually involve redundant datapoints, going over every datapoint in every iteration takes much unnecessary effort.
- Requires the training data to be stored in memory which makes it very memory-inefficient.
- Can't update online because it requires all the data to be available at every update.

- Have to choose learning rate manually. If learning rate is too large, it may escape the true minima. But if learning rate is too small, the learning will be very slow.
- May get stuck in local minima.

Stochastic Gradient Descent(SGD)[**SGD**] improved GD in terms of the number of data points used in every update. In SGD, before every update, a random sample is chosen to calculate the gradient of the loss function and then update the parameter value. The update rule is as follows:

$$\theta_t = \theta_{t-1} - \eta \nabla L(\theta; x_i, y_i) \tag{2}$$

The advantage of SGD are:

- Avoid redundant gradient calculation in every update.
- Can jump to better local minimum when the function is not convex.

The weaknesses of SGD are:

- Hard to set learning rate manually.
- Treat all the features equally and update in the same way.
- Not stable.
- Still not able to get rid of saddle points.

SGDM[**SGDM**] added values to the gradient term when updating parameters. It utilized the idea of ”momentum” to achieve larger decrease when the loss function in this dimension keeps decreasing while smaller decrease when the loss function in this dimension starts to increase. The update rule is as follows:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla L(\theta; x_i, y_i) \\ \theta_t &= \theta_{t-1} - v_t \end{aligned} \tag{3}$$

The advantage of SGDM are:

- Converges to local minima faster.
- Reduces oscillation.
- Able to pass saddle points.

The weaknesses of SGDM are:

- Update all the features equally and in the same way.
- Have to set learning rate manually.

Adagrad[**Duchi**] adopts different learning rates to different features, adapting high learning rate to the parameters associated with infrequently occurring features, and low learning rates for parameters associated with frequent features. This makes sense because it pays more attention to infrequently occurring features and therefore, it is suitable for coping with sparse data. The detailed update rule is as follows:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{ii,t} + \epsilon}} g_{i,t} \tag{4}$$

where $\theta_{t,i}$ represents the i^{th} dimension of parameter θ at timestep t, $G_{t,ii}$ represents the sum squared gradient of the i^{th} dimension of parameter θ until timestep $t (G_t = \sum_{\beta=1}^{t} g_\beta g_\beta^T)$, $g_{i,t}$ represents the gradient of the i^{th} dimension of parameter θ at timestep t. The advantages of this scheme are:

- Treat features differently and adjust the learning rate according to their frequencies. Parameters with higher past gradients will update drastically while parameters with sparse or minor gradients will update slowly.
- No need to set learning rate beforehand.
- Beneficial in training deep neural network because it takes into account different scale of gradients in each layer.

The disadvantages of this scheme is:

- Very sensitive to hyper-parameter(η) selection.
- May lead to premature convergence because with the accumulation of squared gradient, the learning rate becomes smaller and smaller so that the parameters will stop updating.

RMSprop made changes to Adagrad, updating learning rates based on a recent period of gradients instead of all the past gradients. The $G_{ii,t}$ term is now replaced by the recursive term, running average of gradient:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \tag{5}$$

The update rule will be:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \tag{6}$$

Some advantages about RMSprop are:

- No manual setting of a learning rate.
- Separate dynamic learning rate per-dimension.
- Robust to large gradients, noise and architecture choice.

- Applicable in both local or distributed environments
- Get rid of continual decay of learning rate.

Some weaknesses about RMSprop are:

- Sensitive to hyper-parameter(η) selection
- Not able to get rid of saddle points.

Adadelta[**Zeiler**] also updates learning rates based on a recent period of gradients instead of all the past gradients, similar to RMSprop. The $G_{ii,t}$ term is replaced by the running average of gradient:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma)g_t^2 \tag{7}$$

In addition, in order to avoid the mismatch of units in the updated rule of Adagrad and the continual decay of learning rate despite the current learning rate is already very tiny, Adadelta introduced decaying squared parameter updates:

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2 \tag{8}$$

The complete update rule will be:

$$\theta_{t+1} = \theta_t - \frac{\sqrt{\theta_{t-1}+\epsilon}}{\sqrt{G_t+\epsilon}} g_t \tag{9}$$

Some advantages about Adadelta are:

- No manual setting of a learning rate.
- Insensitive to hyper-parameters.
- Separate dynamic learning rate per-dimension.
- Minimal computation over gradient descent.
- Robust to large gradients, noise and architecture choice.
- Applicable in both local or distributed environments.
- Get rid of continual decay of learning rate.

Some weaknesses about Adadelta are:

- Not able to get rid of saddle points.

Adam[**KingmaB14**] is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients G_t like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients g_t, similar to momentum. The calculation of G_t and g_t goes like this:

$$\begin{aligned} v_t &= \beta_1 v_{t-1} + (1-\beta_1) g_t^2 \\ m_t &= \beta_2 m_{t-1} + (1-\beta_2) g_t \end{aligned} \tag{10}$$

And to eliminate bias:

$$\begin{aligned} \hat{v_t} &= \frac{v_t}{1-\beta_1^t} \\ \hat{m_t} &= \frac{m_t}{1-\beta_2^t} \end{aligned} \tag{11}$$

Then, update θ according to this formula:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v_t}} + \epsilon} \hat{m_t} \tag{12}$$

Some advantages about Adam are:

- The magnitudes of parameter updates are invariant to rescaling of the gradient
- Stepsizes are approximately bounded by the stepsize hyper-parameter
- It does not require a stationary objective
- It works with sparse gradients
- It naturally performs a form of step size annealing.

Some weaknesses about Adam are:

- May have the continual decay of learning rate issue like Adagrad.

In summary, full gradient descent and stochastic gradient descent both go towards the direction of decreasing the function. However, without any thick, they are likely to get stuck at saddle points. With momentum, SGDM can pass the saddle point by taking advantage of the gradient inertia. When dataset is sparse, adaptive methods such as Adagrad, Adadelta and RMSprop perform better because they update different parameters using different learning rate. A comprehensive method called Adam utilizes both the momentum trick as well as adaptive idea to achieve better results.

ii)

We used logistic regression to do binary classification and at the same time used different gradient descent algorithms. We used several online dataset to make comparisons. Firstly, we used iris dataset[**iris**]. Our results are shown in Figure 1 and 2.

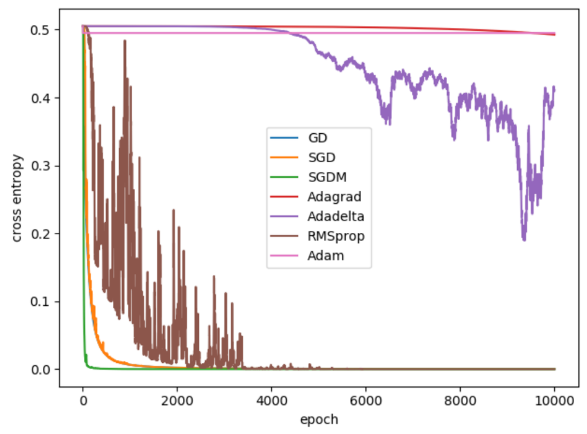

Figure 1: Comparison of seven gradient descent techniques(MSE loss function). Iris dataset(100 data samples, 2 classes, and 4 features each datapoint)

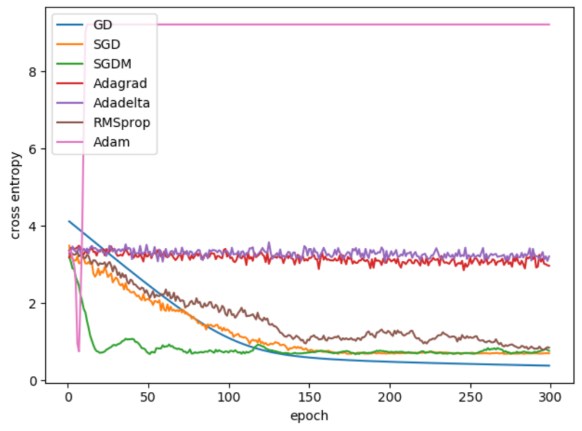

Figure 2: Comparison of seven gradient descent techniques(cross entropy loss function). Iris dataset(100 data samples, 2 classes, and 4 features each datapoint)

Each algorithm starts with initial weight vector $\vec{0}$. From the above figures, we can see that SGD and GD works pretty good and converge fast. While the adaptive algorithms converge pretty slowly and unstable because now the dataset is pretty small and only have four features. In this simple case, SGD and GD outperforms adaptive methods(Adagrad, RMSprop and Adadelta). It is worth notice that Adam doesn't seem to decrease the both the loss functions. It's because the decreasing rate of the gradient of both the two loss functions are so fast that the momentum term outweighs gradient term in updating. So Adam seems to converge early and doesn't arrive at minimum. Next, we used breast dataset to test the techniques when features become 9. The results are as follows:

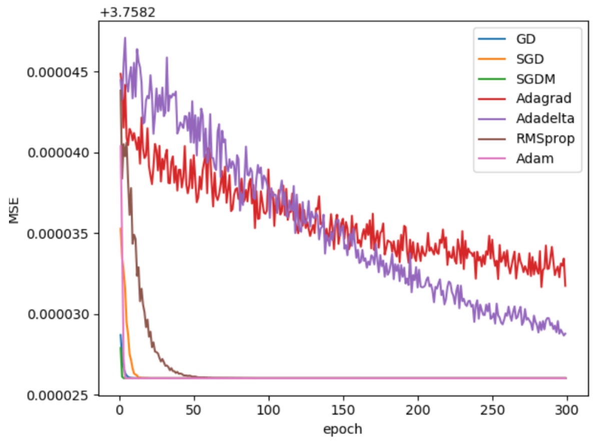

Figure 3: Comparison of seven gradient descent techniques(MSE loss function). Breast cancer dataset(699 data samples, 2 classes, and 9 features each datapoint)

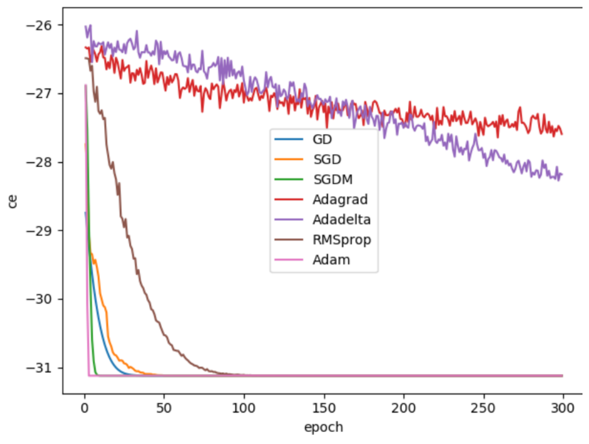

Figure 4: Comparison of seven gradient descent techniques(cross entropy loss function). Breast cancer dataset(699 data samples, 2 classes, and 9 features each datapoint)

From the results above, we see that except the adaptive methods, the other techniques work pretty fast and stable. This means that, still, the small feature space prevents the adaptive methods from performing themselves. To zoom in the first 50 iterations, we get the following result.

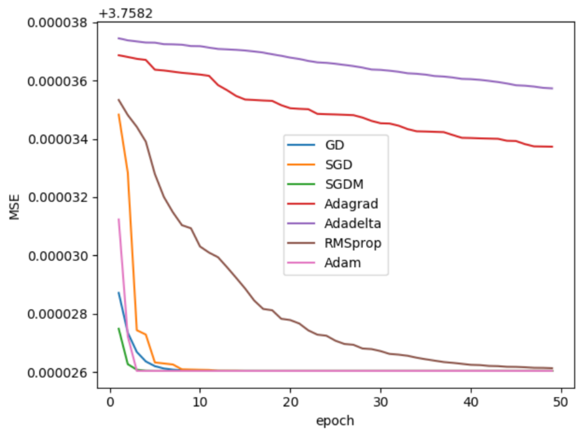

Figure 5: Comparison of seven gradient descent techniques(cross entropy loss function). Breast cancer dataset(zoom in for first 50 iterations)

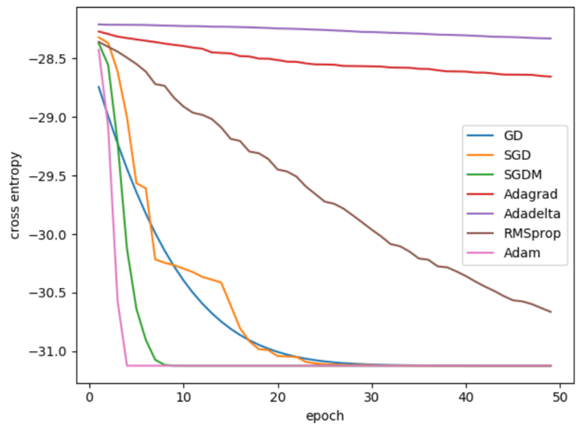

Figure 6: Comparison of seven gradient descent techniques(cross entropy loss function). Breast cancer dataset(zoom in for first 50 iterations)

From the first 50 iterations, we can see that Adagrad, RMSprop and Adadelta still works slowly while others converge really fast. And SGDM and Adam outperms GD and SGD because the gradient now is in normal range (unlike in the previous dataset). And because both loss functions are convex, no techniques get stuck in the local minima.

Problem 2

i)

1. Each w_i starts at 1 and gets promoted on every mistake made on positive example.
2. Each promotion doubles at least one of the relevant weight.
3. Only weights that are $< d$ can be doubled, thus each weight can be doubled at most: $log(d) + 1$.
4. So, the total number of mistakes on positive examples has an upper bound of $M_+ \leq r(1 + log(d))$.

ii)

1. Each w_i starts at 1 and gets demoted on every mistake made on negative example.
2. Each demotion halves at least one irrelevant weight.
3. Before each demotion we had: $w_1x_1 + w_2x_2 + .. + w_dx_d \geq d$
4. So, each mistake on negative example decrease the total weight by at least $\frac{d}{2}$.

iii)

$$0 \leq TW(t) \leq TW(0) + (d)M_+ - (\frac{d}{2})M_-$$

1. Setting $TW(0) = d$, because initially we had all $w_i = 1$, and therefore $\sum w_i = d$

$$\begin{aligned} 0 \leq TW(t) &\leq d + (d)M_+ - (\frac{d}{2})M_- \\ 0 &\leq d + (d)M_+ - (\frac{d}{2})M_- \\ (\frac{d}{2})M_- &\leq d + (d)M_+ \\ (\frac{1}{2})M_- &\leq 1 + M_+ \\ M_- &\leq 2 + 2M_+ \end{aligned}$$

2. Therefore the total number of mistakes this algorithm can make is:

$$\begin{aligned}
&M_+ + M_- \leq M_+ + 2 + 2M_+ \\
&M_+ + M_- \leq 2 + 3M_+ \\
&M_+ + M_- \leq 2 + 3r(1 + log(d))
\end{aligned}$$

Problem 3

https://www.stat.cmu.edu/ cshalizi/mreg/15/lectures/06/lecture-06.pdf
https://www.quantstart.com/articles/Maximum-Likelihood-Estimation-for-Linear-Regression

i)

The generative model for the rating of movie j by user i is:

$$r_{i,j} = u_i \cdot v_j + \epsilon_{i,j} \tag{13}$$

where $\epsilon_{i,j}$ is distributed as independent zero-mean and σ^2-variance Gaussian noise. So $r_{i,j} = N(u_i \cdot v_j, \sigma^2)$. The parameters are u_i, v_j and σ in which u_i are $n \times k$ matrix, v_j are $d \times k$ matrix and σ is a real number. So the total number of parameters is $(n + d) \times k + 1$.

ii)

The maximum likelihood estimation of parameters are:

$$\text{MLE} = \text{arg max}_{u_i \cdot v_j} \prod_{i=1}^{n} \prod_{j=1}^{d} p(r_{i,j} | u_i \cdot v_j) \tag{14}$$

The likelihood above is the objective want to optimize (maximize). There are no constraints other than the domain constrains: u_i and v_j are both k-dimentional vector and $r_{i,j}$ is a real number.

iii)

Using the *negative log likelihood* function, Eq. 14 could be optimized as:

$$\begin{aligned} \text{MLE} &= \text{arg max}_{u_i \cdot v_j} \prod_{i=1}^{n} \prod_{j=1}^{d} p(r_{i,j} | u_i \cdot v_j) \\ &= \text{arg min}_{u_i \cdot v_j} \sum_{i=1}^{n} \sum_{j=1}^{d} - \log p(r_{i,j} | u_i \cdot v_j) \\ &= \text{arg min}_{u_i \cdot v_j} \sum_{i=1}^{n} \sum_{j=1}^{d} (u_i \cdot v_j - r_{i,j})^2 \\ &= \text{arg min}_{u_i \cdot v_j} \sum_{i=1}^{n} \sum_{j=1}^{d} (u_i \cdot v_j)^2 - 2r_{i,j}(u_i \cdot v_j) \end{aligned} \tag{15}$$

We get the optimized form above ignoring terms independent of u_i and v_j, as discussed in Lec 5.

iv)

Yes, the negative log-likelihood problem formulation in part (iii) is a convex optimization problem with respect to the paramter u_i or v_j.

A problem is a convex problem when and only when both function and constraints are convex. As we discussed in part (ii), there is no constraint we need to worry about. So we only need to show that the negative log-likelihood problem formulation in part (iii) is convex with respect to the parameter u_i or v_j. As we could see, the formulation in part (iii) is twice continuously differentiable. So if its Hessian matrix of second partial derivatives with respect to u_i (or v_j) is positive semidefinite, the fuction is convex and hence the optimization problem is convex. [1]

Let's look at u_i first. Since u_i and v_j are both k-dimensional, they could be expressed as $u_i = [u_i^{(1)}, u_i^{(2)}, \cdots, u_i^{(k)}]^{\mathrm{T}}$, $v_j = [v_j^{(1)}, v_j^{(2)}, \cdots, v_j^{(k)}]^{\mathrm{T}}$. Let $F = \sum_{i=1}^{n}\sum_{j=1}^{d}(u_i \cdot v_j)^2 - 2r_{i,j}(u_i \cdot v_j)$. $\frac{\partial F}{\partial u_i} = \sum_{j=1}^{d} 2(u_i \cdot v_j - r_{i,j})v_j$.

$$
\begin{aligned}
\frac{\partial^2 F}{\partial u_i^2} &= \begin{bmatrix} \sum_{j=1}^{d} 2(v_j^{(1)})^2 & \sum_{j=1}^{d} 2v_j^{(1)}v_j^{(2)} \cdots \sum_{j=1}^{d} 2v_j^{(1)}v_j^{(k)} \\ \sum_{j=1}^{d} 2v_j^{(2)}v_j^{(1)} & \sum_{j=1}^{d} 2(v_j^{(2)})^2 \cdots \sum_{j=1}^{d} 2v_j^{(2)}v_j^{(k)} \\ \cdots & \\ \sum_{j=1}^{d} 2v_j^{(k)}v_j^{(1)} & \sum_{j=1}^{d} 2v_j^{(k)}v_j^{(2)} \cdots \sum_{j=1}^{d} 2(v_j^{(k)})^2 \end{bmatrix} \\
&= \sum_{j=1}^{d} 2 \begin{bmatrix} (v_j^{(1)})^2 & v_j^{(1)}v_j^{(2)} \cdots v_j^{(1)}v_j^{(k)} \\ v_j^{(2)}v_j^{(1)} & (v_j^{(2)})^2 \cdots v_j^{(2)}v_j^{(k)} \\ \cdots & \\ v_j^{(k)}v_j^{(1)} & v_j^{(k)}v_j^{(2)} \cdots (v_j^{(k)})^2 \end{bmatrix} \\
&= \sum_{j=1}^{d} 2f_j
\end{aligned} \tag{16}
$$

Where

$$
f_j = \begin{bmatrix} (v_j^{(1)})^2 & v_j^{(1)}v_j^{(2)} \cdots v_j^{(1)}v_j^{(k)} \\ v_j^{(2)}v_j^{(1)} & (v_j^{(2)})^2 \cdots v_j^{(2)}v_j^{(k)} \\ \cdots & \\ v_j^{(k)}v_j^{(1)} & v_j^{(k)}v_j^{(2)} \cdots (v_j^{(k)})^2 \end{bmatrix} \tag{17}
$$

If we could prove that real matrix f_j is positive semidefinite, then $\frac{\partial^2 F}{\partial u_i^2}$, as the sum of f_j is also positive semidefinite. v_j is defined as a k-dimensional vector and $f_j = v_j v_j^{\mathrm{T}} =<$

[1] See the definition of convex function and Hessian matrix in wikipedia: https://en.wikipedia.org/wiki/Convex_function, https://en.wikipedia.org/wiki/Hessian_matrix.

$v_j, v_j >$, so f_j is a Gram matrix and hence is always positive semidefinite. [2]

In conclusion, as a Gram matrix, f_j is positive semidefinite. The sum of f_j, $\frac{\partial^2 F}{\partial u_i^2}$, which is the second partial derivatives of $F = \sum_{i=1}^{n}\sum_{j=1}^{d}(u_i \cdot v_j)^2 - 2r_{i,j}(u_i \cdot v_j)$ with respect to u_i, is also positive semidefinite. So the function F is convex and the optimization is a convex problem with respect to the parameter u_i.

Similarly, for parameter v_j, we have:

$$\begin{aligned}
\frac{\partial^2 F}{\partial v_j^2} &= \begin{bmatrix} \sum_{i=1}^{n} 2(u_i^{(1)})^2 & \sum_{i=1}^{n} 2u_i^{(1)}u_i^{(2)} \cdots \sum_{i=1}^{n} 2u_i^{(1)}u_i^{(k)} \\ \sum_{i=1}^{n} 2u_i^{(2)}u_i^{(1)} & \sum_{i=1}^{n} 2(u_i^{(2)})^2 \cdots \sum_{i=1}^{n} 2u_i^{(2)}u_i^{(k)} \\ \cdots & \\ \sum_{i=1}^{n} 2u_i^{(k)}u_i^{(1)} & \sum_{i=1}^{n} 2u_i^{(k)}u_i^{(2)} \cdots \sum_{i=1}^{n} 2(u_i^{(k)})^2 \end{bmatrix} \\
&= \sum_{i=1}^{n} 2 \begin{bmatrix} (u_i^{(1)})^2 & u_i^{(1)}u_i^{(2)} \cdots u_i^{(1)}u_i^{(k)} \\ u_i^{(2)}u_i^{(1)} & (u_i^{(2)})^2 \cdots u_i^{(2)}u_i^{(k)} \\ \cdots & \\ u_i^{(k)}u_i^{(1)} & u_i^{(k)}u_i^{(2)} \cdots (u_i^{(k)})^2 \end{bmatrix} \\
&= \sum_{i=1}^{n} 2g_i
\end{aligned} \tag{18}$$

Where

$$g_i = \begin{bmatrix} (u_i^{(1)})^2 & u_i^{(1)}u_i^{(2)} \cdots u_i^{(1)}u_i^{(k)} \\ u_i^{(2)}u_i^{(1)} & (u_i^{(2)})^2 \cdots u_i^{(2)}u_i^{(k)} \\ \cdots & \\ u_i^{(k)}u_i^{(1)} & u_i^{(k)}u_i^{(2)} \cdots (u_i^{(k)})^2 \end{bmatrix} \tag{19}$$

Similar prove as before, as a Gram matrix, $g_i = u_i u_i^{\mathrm{T}} =< u_i, u_i >$ is positive semidefinite. The sum of g_i, $\frac{\partial^2 F}{\partial v_j^2}$, which is the second partial derivatives of $F = \sum_{i=1}^{n}\sum_{j=1}^{d}(u_i \cdot v_j)^2 - 2r_{i,j}(u_i \cdot v_j)$ with respect to v_j, is also positive semidefinite. So the function F is convex and the optimization is a convex problem with respect to the parameter v_j.

v)

Yes, the negative log-likelihood problem formulation (in part (iii)) is jointly convex in the parameters u_i and v_j simultaneously.

As discussed in part (iv), there is no constraints on the optimizing objective, we only need to prove the objective function is convex. We could use the definition of

[2]See the definition of Gram matrix in wikipedia: https://en.wikipedia.org/wiki/Gramian_matrix

convex function to prove it, which is: $\beta f(x) + (1-\beta)f(x') \geq f(\beta x + (1-\beta)x')$ for $\beta \in [0,1]$. Here $x = u_i \cdot v_j$ and $x' = u'_i \cdot v'_j$, which are both real numbers. u_i, u'_i, v_j and v'_j are k-dimensional vectors. We choose two sets of parameters (u_i, v_j) and (u'_i, v'_j) to discuss the jointly convex property of the function. Then we have:

$$\begin{aligned} f(x) &= (u_i \cdot v_j)^2 - 2r_{i,j}(u_i \cdot v_j) \\ f(x') &= (u'_i \cdot v'_j)^2 - 2r_{i,j}(u'_i \cdot v'_j) \\ f(\beta x + (1-\beta)x') &= [\beta(u_i \cdot v_j) + (1-\beta)(u'_i \cdot v'_j)]^2 - 2r_{i,j}[\beta(u_i \cdot v_j) + (1-\beta)(u'_i \cdot v'_j)] \end{aligned} \tag{20}$$

Then:

$$\begin{aligned} &\beta f(x) + (1-\beta)f(x') - f(\beta x + (1-\beta)x') \\ =&\beta(u_i \cdot v_j)^2 - 2\beta r_{i,j}(u_i \cdot v_j) + (1-\beta)(u'_i \cdot v'_j)^2 - 2r_{i,j}(1-\beta)(u'_i \cdot v'_j) \\ &- [\beta(u_i \cdot v_j) + (1-\beta)(u'_i \cdot v'_j)]^2 + 2r_{i,j}[\beta(u_i \cdot v_j) + (1-\beta)(u'_i \cdot v'_j)] \\ =&\beta(u_i \cdot v_j)^2 + (1-\beta)(u'_i \cdot v'_j)^2 - [\beta(u_i \cdot v_j) + (1-\beta)(u'_i \cdot v'_j)]^2 \\ =&\beta(u_i \cdot v_j)^2 + (1-\beta)(u'_i \cdot v'_j)^2 - [\beta^2(u_i \cdot v_j)^2 + (1-\beta)^2(u'_i \cdot v'_j)^2 + 2\beta(1-\beta)(u_i \cdot v_j)(u'_i \cdot v'_j)] \\ =&\beta(1-\beta)(u_i \cdot v_j)^2 + \beta(1-\beta)(u'_i \cdot v'_j)^2 - 2\beta(1-\beta)(u_i \cdot v_j)(u'_i \cdot v'_j) \\ =&\beta(1-\beta)(u_i \cdot v_j - u'_i \cdot v'_j)^2 \\ \geq&0 \quad (\text{for} \beta \in [0,1]) \end{aligned} \tag{21}$$

In other words, $\beta f(x) + (1-\beta)f(x') \geq f(\beta x + (1-\beta)x')$ for $\beta \in [0,1]$. And this inequally holds for any choice of u_i and v_j. So the negative log-likelihood problem formulation (in part (iii)) jointly convex in the parameter u_i and v_j simultaneously.

vi)

As we discussed in part (iv), the first derivative of $F = \sum_{i=1}^{n}\sum_{j=1}^{d}(u_i \cdot v_j)^2 - 2r_{i,j}(u_i \cdot v_j)$ with respect to u_i is $\frac{\partial F}{\partial u_i} = \sum_{j=1}^{d} 2(u_i \cdot v_j - r_{i,j})v_j$. Since F is a convex function with respect to u_i, F reaches minimum when $\frac{\partial F}{\partial u_i} = 0$. In other words,

$$\sum_{j=1}^{d} 2(u_i^* \cdot v_j - r_{i,j})v_j = 0 \tag{22}$$

in which u_i^* refers to the optimal setting of u_i. If we consider v_j for j from 1 to d as linearly independent, then we have:

$$u_i^* \cdot v_j - r_{i,j} = 0 \tag{23}$$

which holds for every j value and could be used to get u_i^* if $k = d$. If $k \neq d$, we could not get u_i^* directly from the equation above.

Similarly, we could have the optimal setting of the parameter v_j, v_j^* as following:

$$u_i \cdot v_j^j - r_{i,j} = 0 \tag{24}$$

which holds for every i value and could be used to get v_j^* if $k = n$. If $k \neq n$, we could not get v_j^* directly from the equation above.

vii)

As we discussed in part (i), we predict the rating from Gaussian distribution $r_{i,j} = N(u_i \cdot v_j, \sigma^2)$ in which u_i and v_j are both k-dimensional vectors. If given a new/previously unseen user $\tilde{u}$ or item $\tilde{v}$, we could not give a prediction because we don't know what $\tilde{u}$ or $\tilde{v}$ is.

One way to fix it: we take $\tilde{u}$ as the mean of all known u_i. Then we could give prediction based on the estimated $\tilde{u}$: $r_{\tilde{i},j} = N(\tilde{u} \cdot v_j, \sigma^2) = N(\text{mean}[u] \cdot v_j, \sigma^2)$. Correspondingly, for a new/previously unseen item $\tilde{v}$, take it as the mean of all known v_j and make prediction as: $r_{i,\tilde{j}} = N(u_i \cdot \tilde{v}, \sigma^2) = N(u_i \cdot \text{mean}[v], \sigma^2)$.

Problem 4

i)

The mapping function is:

$$\Phi_{\alpha,B} : x \to (1[B > \gamma > x - \alpha])_{\gamma \in \mathbb{R}}. \tag{25}$$

Let γ denote the mapped vector of x. For any point x, we could always find an infinite dimensional weight vector w to keep the label $\text{sign}(w^T\gamma)$ while doing the feature transformation. Because the γ is infinite dimensional, we can write the dot product of w and γ as a function of γ, namely $w(\gamma)$. Due to the transformation, we have:

$$\text{sign}(\int_{-\infty}^{\infty} w(\gamma)d\gamma) = \text{sign}(\int_{x-\alpha}^{B} w(\gamma)d\gamma) \tag{26}$$

which means only the $w(\gamma)$ corresponding to $\gamma \in (x-\alpha, B)$ are assigned values (otherwise zero) to make $\text{sign}(w^T\gamma)$ positive for x in clase 1 and $\text{sign}(w^T\gamma)$ negative for x in class 0.

We could use induction method to show that the mapping can linearly separate any binary labeling of the n points. If $n = 2$, supposing x_1 is positive, x_2 is negative and $x_1 > x_2$.

We have:

$$\text{sign}(\int_{x_1-\alpha}^{B} w(\gamma)d\gamma) = 1. \tag{27}$$

We could choose $w(\gamma) = 1$ for $\gamma \in (x_1 - \alpha, B)$. Then

$$\begin{aligned} -1 &= \text{sign}(\int_{x_2-\alpha}^{B} w(\gamma)d\gamma) \\ &= \text{sign}(\int_{x_1-\alpha}^{B} w(\gamma)d\gamma + \int_{x_2-\alpha}^{x_1-\alpha} w(\gamma)d\gamma) \\ &= \text{sign}(B - x_1 + \alpha + \int_{x_2-\alpha}^{x_1-\alpha} w(\gamma)d\gamma) \end{aligned} \tag{28}$$

Let $F_1 = (\int_{x_1-\alpha}^{B} w(\gamma)d\gamma) = B - x_1 + \alpha$. We could choose $w(\gamma) = \frac{-2F_1}{x_1-x_2}$ for $\gamma \in (x_2 - \alpha, x_1 - \alpha)$. Because $w(\gamma) < \frac{x_1-B-\alpha}{x_1-x_2}$ for $\gamma \in (x_2 - \alpha, x_1 - \alpha)$, $\text{sign}(\int_{x_2-\alpha}^{B} w(\gamma)d\gamma) = -1$. The transformation of x_1 and x_2 are separable.

If $x_1, \cdots, x_i$ are linearly separable by the mapping with weight vector $w(\gamma)$, the point x_{i+1} has different label with x_i. Let $F_i = (\int_{x_i-\alpha}^{B} w(\gamma)d\gamma)$, we could have $w(\gamma) =$

$\frac{-2F_i}{x_i - x_{i+1}}$ for $\gamma \in (x_{i+1} - \alpha, x_i - \alpha)$. Similarly, we have :

$$\begin{aligned}
\text{sign}(w^T \cdot x_{i+1}) &= \text{sign}(\int_{x_{i+1}-\alpha}^{B} w(\gamma) d\gamma) \\
&= \text{sign}(\int_{x_i-\alpha}^{B} w(\gamma) d\gamma + \int_{x_{i+1}-\alpha}^{x_i-\alpha} w(\gamma) d\gamma) \\
&= \text{sign}(F_i - 2F_i) \\
&= -\text{sign}(F_i) \\
&= -\text{sign}(\int_{x_i-\alpha}^{B} w(\gamma) d\gamma) \\
&= -\text{sign}(w^T \cdot x_i)
\end{aligned} \tag{29}$$

So x_i and x_{i+1} are linearly sepearable. In conclusion, for any n distinct points $x_1, \cdots, x_n \in [-B, B]$, there exists $\alpha > 0$ such that the mapping $\Phi_{\alpha,B}$ can linearly separate any binary labeling of the n points. We only needs to assign the weight vector section by section.

ii)

$$\begin{aligned}
\Phi_{\alpha,B}(x) \cdot \Phi_{\alpha,B}(x') &= \int_{-\infty}^{\infty} 1[B > \gamma > x - \alpha] \cdot 1[B > \gamma > x' - \alpha] \; d\gamma \\
&= \int_{\max(x-\alpha, x'-\alpha)}^{B} 1 \; d\gamma \\
&= B - \max(x, x') + \alpha
\end{aligned} \tag{30}$$

Problem 5

i)

The Second Model we used is the user-based collaborative filtering model [**Kembellec2014**]. This model is a kind of memory based algorithms, this model is based on the idea that similar users share similar ratings. The ratings are predicted using the similarities between users and the user we are trying to predict the ratings for. The algorithm works as follows:

1. calculate the similarity between the users using cosine similarity:

$$\text{RawCosine}(u, v) = \frac{\sum_{k \in I_u \cap I_v} r_{uk} \cdot r_{vk}}{\sqrt{\sum_{k \in I_u \cap I_v} r_{uk}^2} \cdot \sqrt{\sum_{k \in I_u \cap I_v} r_{vk}^2}}$$

Figure 7: Function to calculate cosine

2. Predict the ratings of the user for the movies. By calculating the mean centered rating (the user's actual ratings - mean) multiplied by the similarity which is the weight of this average is equal to the value of the cosine similarity between the user we want to predict the review for and every users, that we calculated in the previous step. the equation for the rating is as follows:

$$\hat{r}_{uj} = \mu_u + \frac{\sum_{v \in P_u(j)} \text{Sim}(u, v) \cdot s_{vj}}{\sum_{v \in P_u(j)} |\text{Sim}(u, v)|} = \mu_u + \frac{\sum_{v \in P_u(j)} \text{Sim}(u, v) \cdot (r_{vj} - \mu_v)}{\sum_{v \in P_u(j)} |\text{Sim}(u, v)|}$$

Figure 8: Function for predicting the rating

This model uses the user's other ratings on the k most similar movies to make the prediction. This model is expected to give more accurate predictions to the users rating, but it may suffer from the sparsity of our training data.

ii)

In Q3, we model the rating of movie j from user i as $r_{i,j} = N(u_i \cdot v_j, \sigma^2)$ which has a $u_i \cdot v_j$-mean and σ^2-variance Gaussian distribution. Parameters u_i, $v_j \in \mathcal{R}^k$ are k-dimensional vectors for i from 1 to n and j from 1 to d and σ^2 is a real number. As suggested in Q3, parameters above could be estimated from minimize the negative log likelihood function. Then we have:

$$\begin{aligned}\sigma^2 &= \text{argmin}_{\sigma^2} \sum_{i=1}^{n}\sum_{j=1}^{d} \frac{1}{2}\log\sigma^2 + \frac{(u_i \cdot v_j - r_{i,j})^2}{2\sigma^2} \\ &= \frac{\sum\limits_{i=1}^{n}\sum\limits_{j=1}^{d}(u_i \cdot v_j - r_{i,j})^2}{\sum\limits_{i=1}^{n}\sum\limits_{j=1}^{d} 1}\end{aligned} \tag{31}$$

Similar to the 1-d Gaussian distribution, we could find the optimal value for σ^2 using the stationary point.

However, as we have shown in Q3 (vi), we could not get the parameters u_i or v_j directly from the stationary point of the negative log likelihood function. Instead, we could use the Gradient Descent method to get to the minimum of $F = \sum\limits_{i=1}^{n}\sum\limits_{j=1}^{d}(u_i \cdot v_j)^2 - 2r_{i,j}(u_i \cdot v_j)$ As we discussed in Q3 (iv), F is convex with respect to the parameter u_i or v_j. Also, as in Q3 (v), F is jointly convex in parameters u_i and v_j. Since F is a convex function, the local minimum we find is in fact the global minimum. As we discussed in Q1, different Gradient Descent methods give similar accuracy for convex functions and converges to the minimum quickly. So we use Stochastic Gradient Descent method for all the u_i and v_j.

Similar to what we did in part (i), we split the whole dataset into two sets: training and testing, which has a ratio of sample sizes as 3 : 1. In the training process, we randomly initialize u_i and v_j vectors and then update u_i and v_j using SGD:

$$\begin{aligned}u_i' &= u_i - \alpha\frac{\partial F}{\partial u_i} \\ v_j' &= v_j - \alpha\frac{\partial F}{\partial v_j}\end{aligned} \tag{32}$$

where the partial derivative of F are:

$$\begin{aligned}\frac{\partial F}{\partial u_i} &= \sum_{j=1}^{d} 2(u_i \cdot v_j - r_{i,j})v_j \\ \frac{\partial F}{\partial v_j} &= \sum_{i=1}^{n} 2(u_i \cdot v_j - r_{i,j})u_i\end{aligned} \tag{33}$$

We get the optimal estimation of u_i and v_j when F reaches minimum, then get corresponding optimal σ^2 from Eq. 31.

In the testing process, we give prediction as $y_l = r_{i,j} = N(u_i \cdot v_j, \sigma^2)$ with u_i, v_j and σ^2 from training. Then we use the mean square error (MSE) to evaluate the performance:

$$MSE = \frac{1}{m}\sum_{l=1}^{m}(y_l - \hat{y}_l) \tag{34}$$

where y_l is the prediction and y_l is the label from testing set.

In the code for SGD method, we have three hyper-parameters: the number of features k in u_i or v_j, the learning rate and the number of iterations.

First, let's look at the number of features. Since there are about 9000 movies and about 600 users, using the conclusion from Q3.(i), we have $(n + d)k + 1 \approx 9600k$ variables. The datafile contains around 100, 000 movie ratings. To avoid redundant features, we should have an upper bound of k that ensures $9600k < 100,000$. In other words, $k < 10$. Considering the training sample is about 3/4 of all the movie ratings, which is about 75, 000 records, the actual k might be even smaller. On the other hand, k should not be too small because the model would be too simplified and the error would be large. So we fixed $\alpha = 0.01$ and 10 iterations and tested $k = [1, 3, 5, 7, 9]$. In Fig. 9, we show the training MSE and testing MSE as the function of k. From this plot, we could see that the training MSE is larger than testing MSE with all k values. The MSE decreases with the increasing k while the MSE lines are quite flat when k increases from 7 to 9. For the running efficiency and based on our previous discussion, we choose $k = 7$.

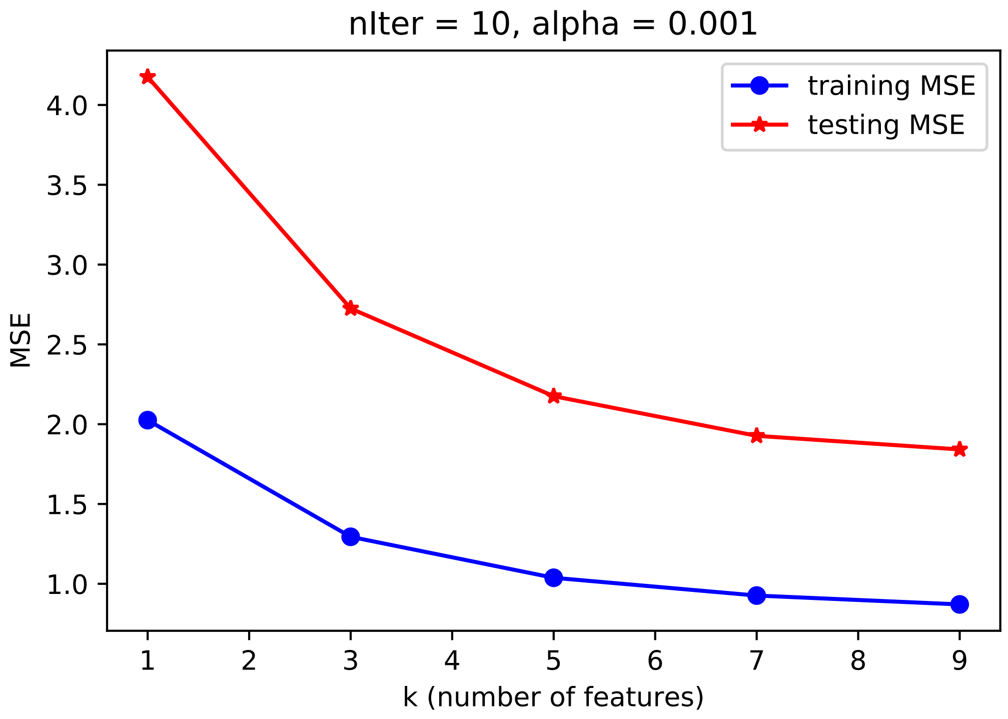

Figure 9: Performance of SGD method as a function of k.

Then, the learning rate. As we discussed in Q1, the learning rate should not be too small, or in limited iterations, it may not descent enough and not reach minimum. On the other hand, it should not be too big, or it may skip the minimum, jumping back and forth. So we fixed $k = 7$ and 10 iterations and tested $\alpha = [0.0005, 0.001, 0.005, 0.01, 0.03]$. In Fig. 10, we show the training MSE and testing

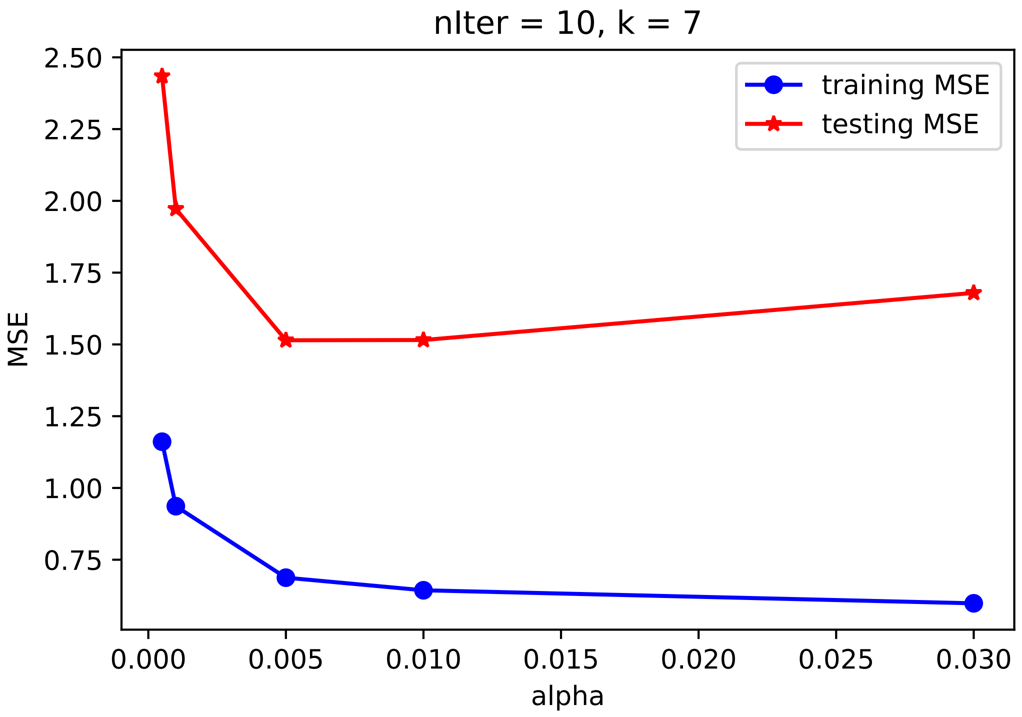

Figure 10: Performance of SGD method as a function of α.

MSE as the function of α. From this plot, we could see that the training MSE is larger than testing MSE with all α values. The training MSE decreases with the increasing α. When α increases from 0.0005 to 0.005, the training MSE drops significantly but when α increases tremendously increases from 0.005 to 0.03, the training MSE is quite flat. The testing MSE shows similar trend while has a slight increase with α increasing from 0.005 to 0.03. So we choose $\alpha = 0.005$.

Finally, the number of iterations. Ideally, we would want to stop the gradient descent when F reaches minimum, which means the partial derivatives are zero. But in reality, it takes a lot of running time which usually seems unnecessary. We don't need F to be minimum, instead we want it to be small enough. So people commonly set a maximum to limit the iteration number. Generally speaking, if we choose proper k and proper α, the MSE would decrease with the increase of iteration number, as shown in Fig. 11. We fixed $k = 7$ and $\alpha = 0.005$ and tested number of iterations = $[10, 50, 100]$.

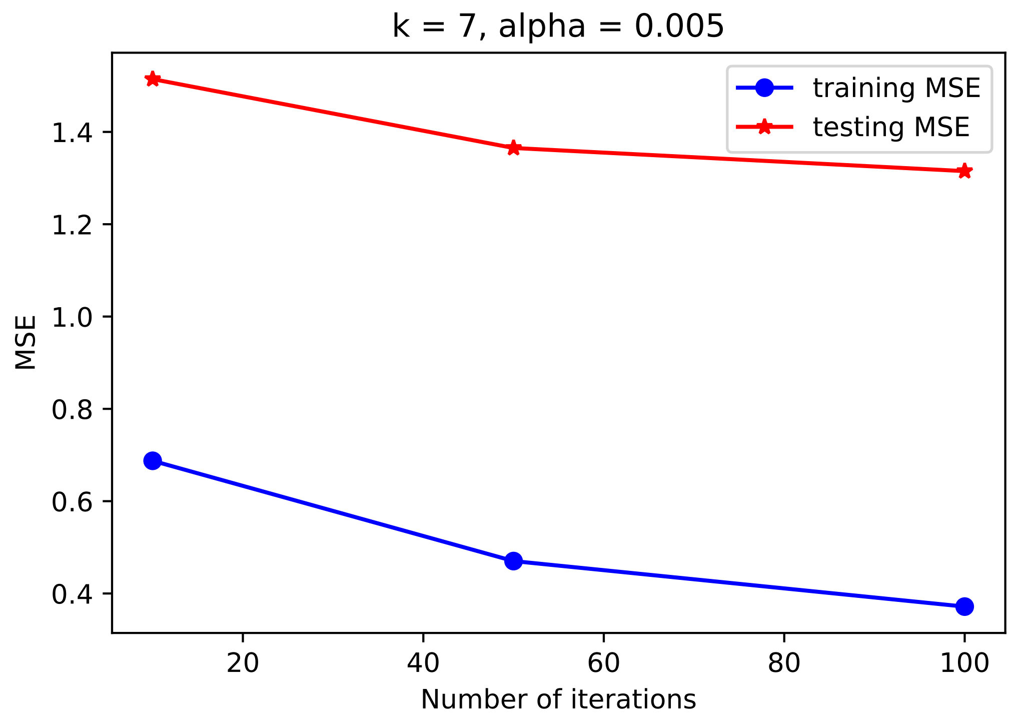

Figure 11: Performance of SGD method as a function of iteration number.

In conclusion, we implement the model in Q3 with u_i and v_j optimized with Stochastic Gradient Descent method (and σ^2 based on u_i and v_j). We choose $k = 7, \alpha = 0.005$ and 100 iterations for 75627 training ratings and 25209 testing ratings. Because of the computation resource limit, we only separated datasets into training and testing sets and tried several parameters. A better way to do it is to use k-fold cross validation to choose the optimal combination of parameters. Also, the parameters should be combined to do parameter search, instead of what we do here, which is fixing 2 parameters and discuss the rest one parameter.

iii)

The second model from Q3, is better for the movie recommendation data. Because the data suffers from sparsity, and this effects the results of the collaborative filtering because it is not robust to data sparsity.

The performance for various splits of data is going to be different because we rely heavily on the training data. Furthermore, the models give better results when trained on larger dense training data-set.
Both models will give the new user/ new movie the average of the previous ratings, so

it will recommend to the user the most popular movies.