CS 7641 CSE/ISYE 6740 Homework 4

Luis M. Perez (902896492)

November 4, 2013

- I discussed the questions with Sari Wolmer

1 Information Theory

Suppose the joint probability distribution of two binary random variables X and Y are given as follows.

$X \backslash Y$	0	1
0	$\frac{1}{3}$	$\frac{1}{3}$
1	0	$\frac{1}{3}$

(a) Find entropy $H(X)$ and $H(Y)$. [6 pts]

First, we compute the marginal distributions:

$$P(X=0) = P(X=0,Y=0) + P(X=0,Y=1) = \frac{1}{3} + \frac{1}{3} = \frac{2}{3}$$
$$P(X=1) = P(X=1,Y=0) + P(X=1,Y=1) = 0 + \frac{1}{3} = \frac{1}{3}$$
$$P(Y=0) = P(X=0,Y=0) + P(X=1,Y=0) = \frac{1}{3} + 0 = \frac{1}{3}$$
$$P(Y=1) = P(X=0,Y=1) + P(X=1,Y=1) = \frac{1}{3} + \frac{1}{3} = \frac{2}{3}$$

Then,

$$\begin{aligned} H(X) &= -\sum_{i}^{N} P(X=i)\log_2 P(X=i) \\ &= -P(X=0)\log_2 P(X=0) - P(X=1)\log_2 P(X=1) \\ &= -\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3} = 0.92 \end{aligned}$$

$$\begin{aligned} H(Y) &= -\sum_{i}^{N} P(Y=i)\log_2 P(Y=i) \\ &= -P(Y=0)\log_2 P(Y=0) - P(Y=1)\log_2 P(Y=1) \\ &= -\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} = 0.92 \end{aligned}$$

(b) Find conditional entropy $H(X|Y)$ and $H(Y|X)$. [6 pts]

$$\begin{aligned}H(X|Y) &= -P(X=0,Y=0)\log_2\frac{P(X=0,Y=0)}{P(Y=0)} - P(X=0,Y=1)\log_2\frac{P(X=0,Y=1)}{P(Y=1)} \\ &- P(X=1,Y=0)\log_2\frac{P(X=1,Y=0)}{P(Y=0)} - P(X=1,Y=1)\log_2\frac{P(X=1,Y=1)}{P(Y=1)} \\ &= -\frac{1}{3}\log_2 1 - \frac{1}{3}\log_2\frac{1}{2} - 0 - \frac{1}{3}\log_2\frac{1}{2} = -\frac{2}{3}\log_2\frac{1}{2} = \frac{2}{3}\end{aligned}$$

$$\begin{aligned}H(Y|X) &= -P(X=0,Y=0)\log_2\frac{P(X=0,Y=0)}{P(X=0)} - P(X=0,Y=1)\log_2\frac{P(X=0,Y=1)}{P(X=0)} \\ &- P(X=1,Y=0)\log_2\frac{P(X=1,Y=0)}{P(X=1)} - P(X=1,Y=1)\log_2\frac{P(X=1,Y=1)}{P(X=1)} \\ &= -\frac{1}{3}\log_2\frac{1}{2} - \frac{1}{3}\log_2\frac{1}{2} - 0 - \frac{1}{3}\log_2 1 = -\frac{2}{3}\log_2\frac{1}{2} = \frac{2}{3}\end{aligned}$$

(c) Find mutual information $I(X;Y)$. [4 pts]

$$I(X;Y) = H(X) - H(X|Y) = 0.92 - \frac{2}{3} = 0.25$$

(d) Find joint entropy $H(X,Y)$. [4 pts]

$$H(X,Y) = H(X) + H(Y) - I(X;Y) = 0.92 * 2 - 0.25 = 1.59$$

2 Linear Regression

In class, we derived a closed form solution (normal equation) for linear regression problem: $\theta = (X^TX)^{-1}X^TY$. A probabilistic interpretation of linear regression tells us that we are relying on an assumption that each data point is actually sampled from a linear hyperplane, with some white noise. The noise follows a zero-mean Gaussian distribution with constant variance. Mathematically,

$$Y^i = \theta^T X^i + \epsilon^i \tag{1}$$

where $\epsilon^i \sim \mathcal{N}(0, \sigma^2 I)$. In other words, we are assuming that each data point is independent to each other (no covariance) and that each data point has same variance.

(a) Using the normal equation, derive the expectation $\mathbb{E}[\theta]$. [5 pts]

$$Y = X\theta + \epsilon \Rightarrow E[Y] = X\theta + E[\epsilon] = X\theta$$

Having this result,

$$\begin{aligned}E[\theta] &= E[(X^TX)^{-1}X^TY] \\ &= (X^TX)^{-1}X^TE[Y] \\ &= (X^TX)^{-1}(X^TX)\theta \\ &= \theta\end{aligned}$$

(b) Using the normal equation, derive the variance Var$[\theta]$. [5 pts]

$$Y = X\theta + \epsilon \Rightarrow Var[Y] = Var[X\theta] + Var[\epsilon] = 0 + \sigma^2 I = \sigma^2 I$$

Having this result,

$$\begin{aligned}
Var[\theta] &= Var[(X^T X)^{-1} X^T Y] \\
&= (X^T X)^{-1} X^T Var[Y]((X^T X)^{-1} X^T)^T \\
&= (X^T X)^{-1} X^T Var[Y] X (X^T X)^{-1} \\
&= (X^T X)^{-1} X^T \sigma^2 I X (X^T X)^{-1} \\
&= (X^T X)^{-1} X^T \sigma^2 I X (X^T X)^{-1} \\
&= \sigma^2 (X^T X)^{-1} (X^T X)(X^T X)^{-1} \\
&= \sigma^2 (X^T X)^{-1}
\end{aligned}$$

(c) Under the white noise assumption above, someone claims that θ follows Gaussian distribution with mean and variance in (a) and (b), respectively. Do you agree with this claim? Why or why not? [5 pts]

Yes, because the θ is defined as a function of X which is fixed and Y which is normally distributed (under the white noise assumption). Thus, θ will follow a Gaussian distribution as Y.

(d) Weighted linear regression

Suppose we keep the independence (no covariance) assumption but remove the same variance assumption. Then, data points would be still sampled independently, but now they may have different variance σ_i. Thus, the covariance matrix would be still diagonal, but with different values:

$$\Sigma = \begin{bmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_n^2 \end{bmatrix}. \tag{2}$$

Derive the normal equation for this problem using matrix-vector notations with Σ. [10 pts]

$$\min Z = (Y - X\theta)^T \Sigma^{-1} (Y - X\theta)$$

$$\begin{aligned}
\frac{\partial Z}{\partial \theta} &= 2X^T \Sigma^{-1}(Y - X\theta) = 0 \\
&\Rightarrow X^T \Sigma^{-1}(Y - X\theta) = 0 \\
&\Rightarrow X^T \Sigma^{-1} Y - X^T \Sigma^{-1} X\theta = 0 \\
&\Rightarrow X^T \Sigma^{-1} Y = X^T \Sigma^{-1} X\theta \\
&\Rightarrow \hat{\theta} = (X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} Y
\end{aligned}$$

3 Classification Methods

(a) With AdaBoost, we combine many weak learners to output final classification result. For those weak learners, we mentioned that a very simple form such as decision stump (a decision tree with only one node) is enough if its performance is better than random guess. What if, however, we use weak learners which are slightly worse than random guess, that is, a classifier achieving less than 50% of accuracy? Do you think AdaBoost still works? Explain why. [6 pts]

AdaBoost updates the weighting coefficients by:

$$w_n^{m+1} = w_n^m \exp\{\alpha_m I(y_m(x_n) \neq t_n)\}$$

With

$$\alpha_m = \ln\left\{\frac{1-\epsilon_m}{\epsilon_m}\right\}$$

Where ϵ_m is the accuracy rate. By noting that:

- When $\epsilon_m = 0.5$, $\alpha_m = 0$ and there is no update in the weighting coefficients (w_n).
- When $\epsilon_m > 0.5$, $\alpha_m < 0$ and the weighting coefficients will get closer to 0 as the accuracy increases
- When $\epsilon_m < 0.5$, $\alpha_m > 0$ and the weighting coefficients will increase as the accuracy decreases.

The third bullet corresponds to the question asked. It is easy to note that as the accuracy is close to 0, then $\alpha_m \to \infty$. Thus, the weighting coefficients will also tend to infinity making the AdaBoost fail.

(b) It is sometimes argued that logistic regression and SVM are more resistant to outliers than boosting. Do you agree with this claim? Explain why it is correct or incorrect. [6 pts]

Hint: Think about the loss function that each method minimizes. Which loss function do you think would be more reliable for outliers?

Figure 1 plots the loss function for logistic regression, SVM and AdaBoost. The later minimizes an exponential error function. On the left side of the plot it can be observed that the AdaBoost's loss function is considerable higher for large values. Since outliers occur on extreme values, it can be noted that the loss function for these values will be significant large for AdaBoost. Therefore, AdaBoost is more sensitive to outliers than logistic regression and SVM.

(c) Mark T if the statement is true, and F otherwise. Explain why in 1-2 sentences. [8 pts]

- The basic decision tree method learns a decision boundary which is always axis-aligned.

  TRUE. The decision boundaries are orthogonal to one axis of the space. This implies that it is parallel to all other axes and therefore axis-aligned.

- Artificial neural networks can be used both for regression and classification.

  TRUE. Artificial neural networks are used in topics such as time series or function approximation which rely on regression analysis. In addition, it can be used to construct classifiers (i.e., for classification).

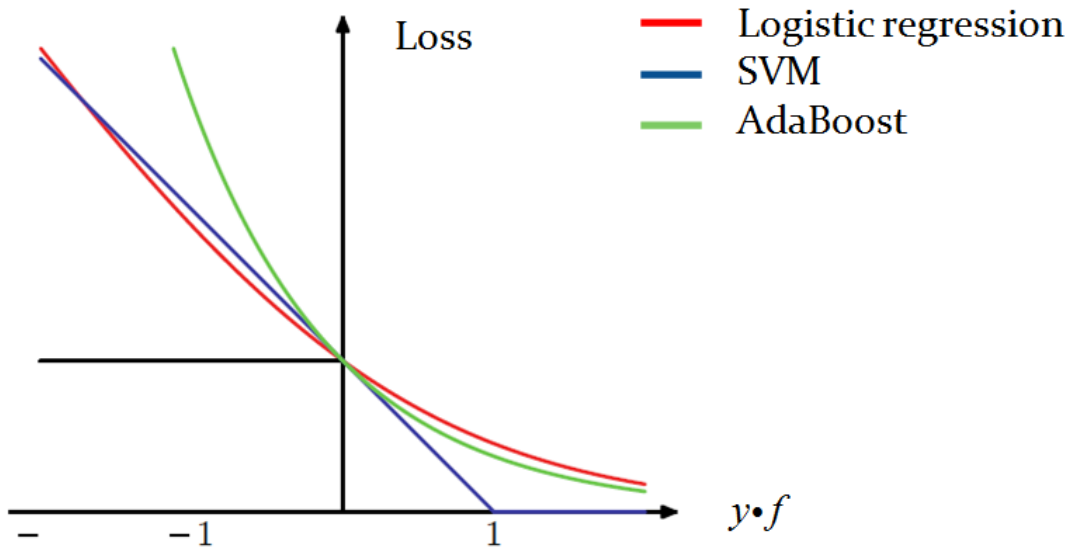

Figure 1: Loss functions for SVM, Regression and AdaBoost. Taken from http://www.stat.ucla.edu/ ywu/AB/adjust/LossFunctions.png

4 Programming: Recommendation System

Personalized recommendation systems are used in a wide variety of applications such as electronic commerce, social networks, web search, and more. Machine learning techniques play a key role to extract individual preference over items. In this assignment, we explore this popular business application of machine learning, by implementing a simple matrix-factorization-based recommender using gradient descent.

Suppose you are an employee in Netflix. You are given a set of ratings (from one star to five stars) from users on many movies they have seen. Using this information, your job is implementing a personalized rating predictor for a given user on unseen movies. That is, a rating predictor can be seen as a function $f : \mathcal{U} \times \mathcal{I} \to \mathbb{R}$, where $\mathcal{U}$ and $\mathcal{I}$ are the set of users and items, respectively. Typically the range of this function is restricted to between 1 and 5 (stars), which is the the allowed range of the input.

Now, let's think about the data representation. Suppose we have m users and n items, and a rating given by a user on a movie. We can represent this information as a form of matrix, namely rating matrix M. Suppose rows of M represent users, while columns do movies. Then, the size of matrix will be $m \times n$. Each cell of the matrix may contain a rating on a movie by a user. In $M_{15,47}$, for example, it may contain a rating on the item 47 by user 15. If he gave 4 stars, $M_{15,47} = 4$. However, as it is almost impossible for everyone to watch large portion of movies in the market, this rating matrix should be very sparse in nature. Typically, only 1% of the cells in the rating matrix are observed in average. All other 99% are missing values, which means the corresponding user did not see (or just did not provide the rating for) the corresponding movie. Our goal with the rating predictor is estimating those missing values, reflecting the user's preference learned from available ratings.

Our approach for this problem is matrix factorization. Specifically, we assume that the rating matrix M is a low-rank matrix. Intuitively, this reflects our assumption that there is only a small number of factors (e.g, genre, director, main actor/actress, released year, etc.) that determine like or dislike. Let's define r as the number of factors. Then, we learn a user profile $U \in \mathbb{R}^{m \times r}$ and an item profile $V \in \mathbb{R}^{n \times r}$. (Recall that m and n are the number of users and items, respectively.) We want to approximate a rating by an inner product of two length r vectors, one representing user profile and the other item profile. Mathematically, a

rating by user u on movie i is approximated by

$$M_{u,i} \approx \sum_{k=1}^{r} U_{u,k} V_{i,k}. \tag{3}$$

We want to fit each element of U and V by minimizing squared reconstruction error over all training data points. That is, the objective function we minimize is given by

$$E(U,V) = \sum_{(u,i)\in M} (M_{u,i} - U_u^T V_i)^2 = \sum_{(u,i)\in M} (M_{u,i} - \sum_{k=1}^{r} U_{u,k} V_{i,k})^2 \tag{4}$$

where U_u is the uth row of U and V_i is the ith row of V. We observe that this looks very similar to the linear regression, which we learned in the class (from slide 24 in lecture 15). Recall that we minimize in linear regression:

$$E(\theta) = \sum_{i=1}^{m} (Y^i - \theta^T x^i)^2 = \sum_{i=1}^{m} (Y^i - \sum_{k=1}^{r} \theta_k x_k^i)^2 \tag{5}$$

where m is the number of training data points. Let's compare (4) and (5). $M_{u,i}$ in (4) corresponds to Y^i in (5), in that both are the observed labels. $U_u^T V_i$ in (4) corresponds to $\theta^T x^i$ in (5), in that both are our estimation with our model. The only difference is that both U and V are the parameters to be learned in (4), while only θ is learned in (5). This is where we personalize our estimation: with linear regression, we apply the same θ to any input x^i, but with matrix factorization, a different profile U_u are applied depending on who is the user u.

As U and V are interrelated in (4), there is no closed form solution, unlike linear regression case. Thus, we need to use gradient descent:

$$U_{u,k} \leftarrow U_{u,k} - \mu \frac{\partial E(U,V)}{\partial U_{u,k}}, \qquad V_{i,k} \leftarrow V_{i,k} - \mu \frac{\partial E(U,V)}{\partial V_{i,k}}, \tag{6}$$

where μ is a hyper-parameter deciding the update rate. It would be straightforward to take partial derivatives of $E(U,V)$ in (4) with respect to each element $U_{u,k}$ and $V_{i,k}$. Then, we update each element of U and V using the gradient descent formula in (6).

(a) Derive the update formula in (6) by solving the partial derivatives. [10 pts]

Define

$$e_{u,i} = M_{u,i} - \sum_{k=1}^{r} U_{u,k} V_{i,k} \tag{7}$$

$$\frac{\partial E(U,V)}{\partial U_{u,k}} = -2 \sum_{i|(u,i)\in M} (e_{u,i} * V_{i,k})$$
$$\frac{\partial E(U,V)}{\partial V_{i,k}} = -2 \sum_{u|(u,i)\in M} (e_{u,i} * U_{i,k})$$

Then, the update formulas are

$$U_{u,k} \leftarrow U_{u,k} + 2\mu \sum_{i|(u,i)\in M} (e_{u,i} * V_{i,k})$$
$$V_{i,k} \leftarrow V_{i,k} + 2\mu \sum_{u|(u,i)\in M} (e_{u,i} * U_{i,k})$$

(b) To avoid overfitting, we usually add regularization terms, which penalize for large values in U and V. Redo part (a) using the regularized objective function below. [5 pts]

$$E(U,V) = \sum_{(u,i)\in M} (M_{u,i} - \sum_{k=1}^{r} U_{u,k}V_{i,k})^2 + \lambda \sum_{u,k} U_{u,k}^2 + \lambda \sum_{i,k} V_{i,k}^2$$

(λ is a hyper-parameter controlling the degree of penalization.)

$$\frac{\partial E(U,V)}{\partial U_{u,k}} = -2 \left(\sum_{i|(u,i)\in M} (e_{u,i} * V_{i,k}) \right) + 2\lambda U_{u,k}$$

$$\frac{\partial E(U,V)}{\partial V_{i,k}} = -2 \left(\sum_{u|(u,i)\in M} (e_{u,i} * U_{i,k}) \right) + 2\lambda V_{i,k}$$

Then, the update formulas are

$$U_{u,k} \leftarrow U_{u,k} + 2 \left(\sum_{i|(u,i)\in M} (e_{u,i} * V_{i,k}) \right) - 2\lambda U_{u,k}$$

$$V_{i,k} \leftarrow V_{i,k} + 2 \left(\sum_{u|(u,i)\in M} (e_{u,i} * U_{i,k}) \right) - 2\lambda V_{i,k}$$

(c) Implement `myRecommender.m` by filling the gradient descent part.

You are given a skeleton code `myRecommender.m`. Using the training data `rateMatrix`, you will implement your own recommendation system of rank `lowRank`. The only file you need to edit is `myRecommender.m`. In the gradient descent part, repeat your update formula in (b), observing the average reconstruction error between your estimation and ground truth in training set. You need to set a stopping criteria, based on this reconstruction error as well as the maximum number of iterations. You should play with several different values for μ and λ to make sure that your final prediction is accurate.

Formatting information is here:

Input

- **rateMatrix**: training data set. Each row represents a user, while each column an item. Observed values are one of $\{1, 2, 3, 4, 5\}$, and missing values are 0.
- **lowRank**: the number of factors (dimension) of your model. With higher values, you would expect more accurate prediction.

Output

- **U**: the user profile matrix of dimension user count $\times$ low rank.
- **V**: the item profile matrix of dimension item count $\times$ low rank.

Evaluation [10 pts]

You may have noticed that the code prints both training and test error, in RMSE (Root Mean Squared Error), defined as follows:

$$\sum_{(u,i)\in M} (M_{u,i} - f(u,i))^2$$

where $f(u, i)$ is your estimation, and the summation is over the training set or testing set, respectively. For our grading, we will use another set-aside testing set, which is not released to you. If you observe your test error is less than 1.00 without cheating (that is, directly referring to the test set), you may expect to see the similar performance on the unseen test set as well.

Hand-in [10 pts]

Upload your `myRecommender.m` implementation file. (Do not copy and paste your code in your report. Be sure to upload your `myRecommender.m` file.)

In your report, show the performance (RMSE) both on your training set and test set, with varied `lowRank`. (The default is set to 1, 3, and 5, but you may want to vary it further.) Discuss what you observe with varied low rank. Also, briefly discuss how you decided your hyper-parameters (μ, λ).

Note

- Do not print anything in your code (e.g, iteration 1 : err=2.4382) in your final submission.
- Submit your code with the best parameters you found. We will grade without modifying your code. (Applying cross-validation to find best parameters is fine, though you do not required to do.)
- Please be sure that your program finishes within a fixed number of iterations. Always think of a case where your stopping criteria is not satisfied forever. This can happen anytime depending on the data, not because your code is incorrect. For this, we recommend setting a maximum number of iteration in addition to other stopping criteria.

Solution

Table 4 shows the RMSE values for both train and test data, over different `lowRank` values.

Set / lowRank	1	3	5	7	10
Train	0.9317	0.9257	0.9083	0.9028	0.9026
Test	0.9604	0.9598	0.9515	0.9498	0.9504

Table 1: Summary of performance for different `lowRank` values

- It is observed that better results are achieved with larger `lowRank` values. However, diminishing improvements are observed as well.
- The improvements are higher on the training set. In the test set the improvement is smaller which indicates that it may not be worth having larger number of dimensions to obtain good performance.
- In addition, it is important to note that as we increase the dimensionality (`lowRank` value) more iterations may be needed to 'converge'. This implies a higher running time.
- The algorithm stops when 50 iterations are reached or there is an improvement in the objective function value below 0.1%. Whatever occurs first.

By fixing the learning rate parameter and running the same instance of the problem with different regularizer values, we can obtain a good value for the regularizer (the one that provided the best fit of the data). It is important to note that since the regularizer is not a function of the fixed parameter there is no major issue in tuning the regularizer through this procedure. Figure 3 shows the performance by varying the `regularizer` parameter. It is observed that there are not significant difference in the tested range. A value of `regularizer=1` was chosen as it penalizes over fitting (unlike,`regularizer=0`) and it does not impact

However, since the initial U and V values are obtained randomly it is possible that the performance of the a regularizer value varies depending on the initialization. In order to reduce this potential problem, the procedure was executed for two different instances. In both executions a value of `learningRate = 0.0006` was found to be the best (this does not imply that it is the best overall, it simply implies that it is a 'good' value of the parameter). The Figure 2 shows the result of one execution where it can be observed that the line corresponding to `learningRate = 0.0006` achieves the best result on every iteration.

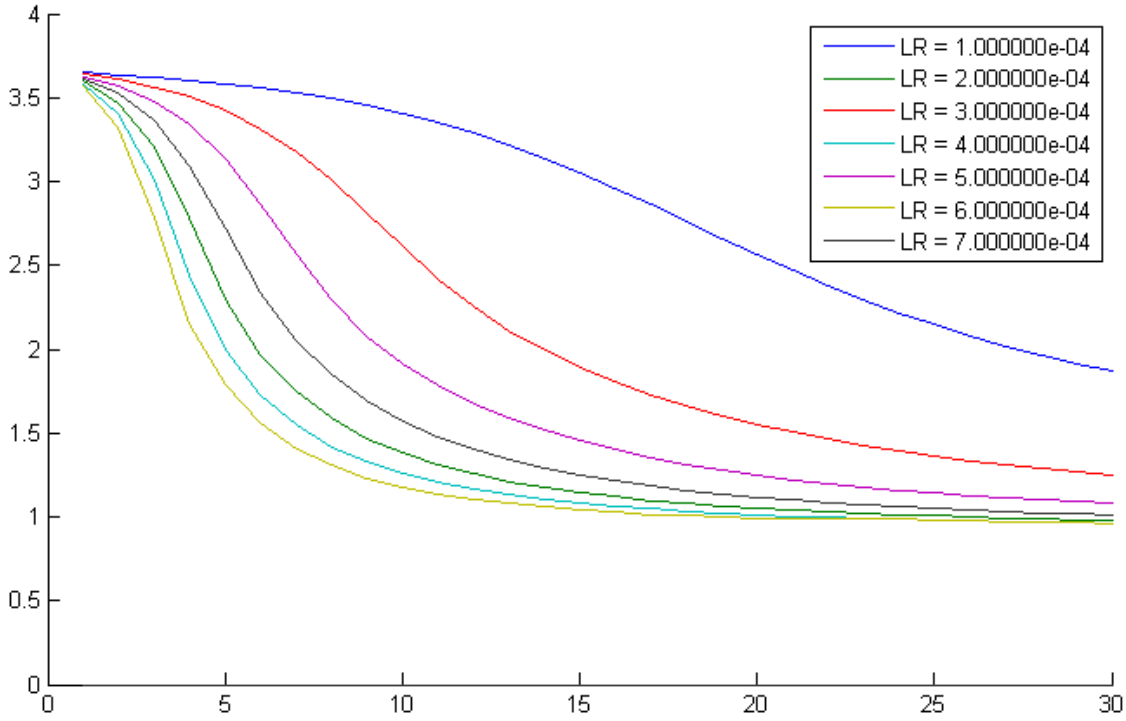

Figure 2: Performance with different values of `learningRate` under fixed `regularizer` value.

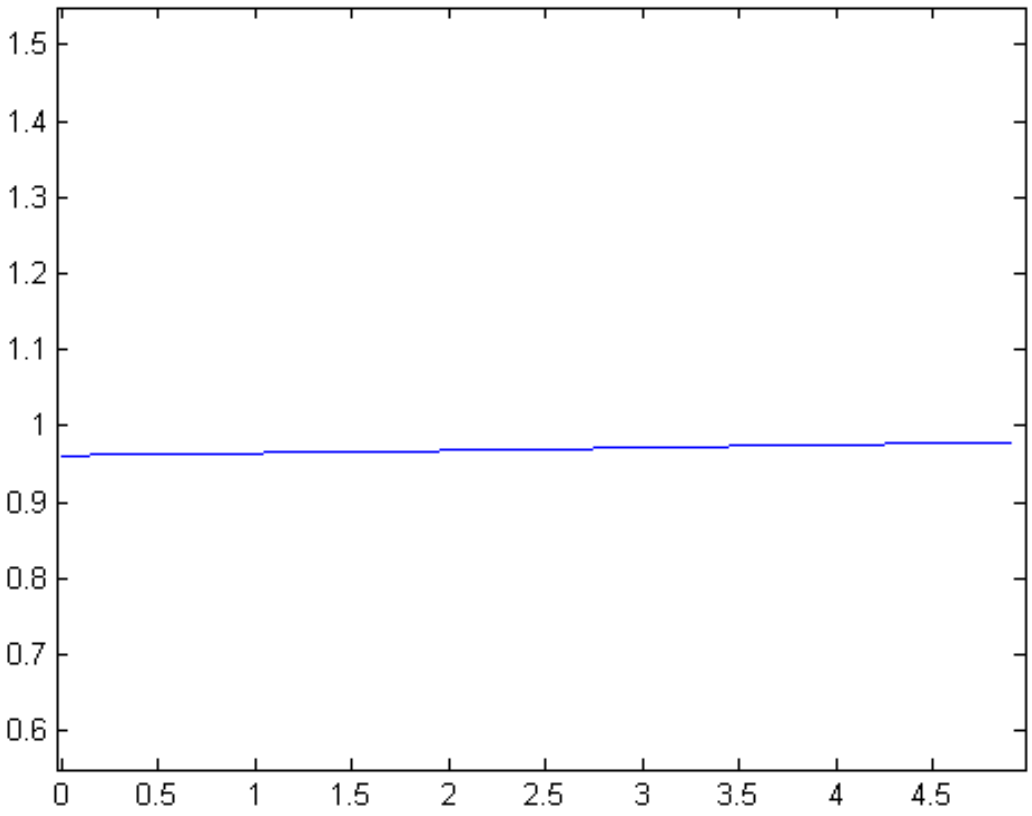

Figure 3: Performance with different values of `regularizer` under fixed `learningRate` value.