DS-GA 1013 Project

Storm Ross

December 2017

1 Introduction

In this report I will be evaluating the performance of a particular method of matrix completion via matrix factorization known as Alternating Least Squares (ALS) and then comparing it to the performance of modified version of the ALS algorithm that I have developed. Matrix factorization methods are typically used for recommendation systems which were one set of methods I utilized while working on a previous report on developing a recommendation system using video game playtime as an implicit rating . While doing so an misinterpretation of the updating rules for ALS algorithm which produced significantly worse results when compared to the other models in the report. This error led to the development of my modification which even after the correction to the updating rules for ALS algorithm showed improved performance on holdout sets.

2 Matrix Factorization

When it comes to matrix completion via matrix factorization the methods that are often used to try to find a solution of the form:

$$R \approx HG$$

Where $R \in \mathbb{R}^{m \times n}$ is the matrix that is try to be estimated, $H \in \mathbb{R}^{m \times k}$, $G \in \mathbb{R}^{k \times n}$, and k is the rank of matrix R. Most methods make the assumption that R has the has low rank of k. The two most often used methods for finding these factor matrices H, G are ALS and Stochastic Gradient Descent (SGD). For both of these methods the metric that is often used to evaluate the predictive performance of the factor matrices , H and G, is Mean Square Error (MSE) between the entries of their product and observed entries of R, R^* where unobserved entries are set to zero. The objective function for these methods involve a term for the MSE while making sure to avoid overfitting to R^* with the addition of a regularization term for each factor matrix [3]:

$$L\left(H,\ G\right) = \frac{1}{N} \sum_{i,j:R^*_{i,j} \neq 0} \left(R^*_{i,j} - H_{i,:}G_{:,j}\right)^2 + \beta \sum_{i=1}^{m} n_{i,:} \left\|H_{i,:}\right\|^2 + \beta \sum_{j=1}^{n} n_{:,j} \left\|G_{:,j}\right\|^2$$

Where β is the regularization parameter for H and G and $n_{i,:}$ and $n_{:,j}$ are the number of non-zero elements in $R^*_{i,:}$ and $R^*_{:,j}$ respectively.

2.1 Regular Alternating Least Squares

The general motivation and idea behind ALS is to alternate between solving for H and G by using the least squares solution to solve for one factor matrix while holding the other one constant. Doing so leads to the following update rules [1]:

$$H_{i,:} = R^*_{i,:} \left(G^{R^*_{i,:}}\right)^T \left(G^{R^*_{i,:}} \left(G^{R^*_{i,:}}\right)^T + n_{i,:}\beta I\right)^{-1}$$

$$G^T_{:,j} = \left(R^*_{:,j}\right)^T H^{R^*_{:,j}} \left(\left(H^{R^*_{:,j}}\right)^T H^{R^*_{:,j}} + n_{:,j}\beta I\right)^{-1}$$

Where $G^{R^*_{i,:}}$ is a copy of G with every column set to zero except those that are non-zero in $r_{i,:}$ and $H^{R^*_{:,j}}$ is a copy of H with every row set to zero except those that are non-zero in $r_{:,j}$. These update rules are repeated until convergence in MSE is reached. Now that the ALS update rules have been defined we can formally describe the algorithm as follows:

1. Initialize $H \in \mathbb{R}^{m \times k}$, $G \in \mathbb{R}^{k \times n}$ randomly
2. Set the first column of G equal to the mean of the observed ratings R^* along the rows and set the first row of H equal to the mean of the observed ratings R^* along the columns
3. Repeat until convergence:
   (a) For $i \in (1, \ldots, m)$
      i. $H_{i,:} = R^*_{i,:} \left(G^{R^*_{i,:}}\right)^T \left(G^{r^*_{i,:}} \left(G^{R^*_{i,:}}\right)^T + n_{i,:}\beta I\right)^{-1}$
   (b) For $j \in (1, \ldots, n)$
      i. $G^T_{:,j} = \left(R^*_{:,j}\right)^T H^{R^*_{:,j}} \left(\left(H^{R^*_{:,j}}\right)^T H^{R^*_{:,j}} + n_{:,j}\beta I\right)^{-1}$

2.2 Modification of ALS: Repeating Alternating Least Squares

The modification for ALS that I am proposing tries to mimic and capture some of the properties of the Singular Value Decomposition of R. The property that this modification is trying to emphasize is the fact that if R is of rank k then only the first k left/right singular vectors and singular values would be needed to approximate R and the first left and right singular vectors should contribute more than or at least as much as the second left and right singular vectors to the approximation of R. The way my modification, Repeating Alternating Least Squares (RALS), does this is that it tries to solve for the first column/row

of H/G using the ALS updating rules from above. After trying its best to approximate R^* with the one column H and one row G the next step is to subtract off the predictions made by the product between one column H and one row G for the observed elements of R^* and this difference becomes the new R^*. With this new R^* the steps of solving for a one column H and a one row G is repeated up to k times with each of these one column Hs and one row Gs. This algorithm can be formally described as:

1. For $f \in (1, \ldots, k)$
   (a) Initialize $G_{f,:}$ to the mean of the current R^* along the rows and Initialize $H_{:,f}$ to the mean of the current R^* along the columns
   (b) Repeat until convergence
      i. For $i \in (1, \ldots, m)$
         A. $H_{i,f} = R^*_{i,:}\left(G^{R^*_{i,:}}_{f,:}\right)^T \left(\left\|G^{R^*_{i,:}}_{f,:}\right\|^2 + n_{i,:}\beta\right)^{-1}$
      ii. For $j \in (1, \ldots, n)$
         A. $G^T_{f,j} = \left(R^*_{:,j}\right)^T H^{R^*_{:,j}}_{:,f} \left(\left\|H^{R^*_{:,j}}_{:,f}\right\|^2 + n_{:,j}\beta\right)^{-1}$
   (c) $R^*_{i,j} \leftarrow R^*_{i,j} - H_{i,:}G_{:,j}\ \forall\ (i,\ j) : R^*_{i,j} > 0$

Where $H^{r^*_{:,j}}_{:,f}$ and $G^{r^*_{i,:}}_{f,:}$ are the fth column and row of $H^{r^*_{:,j}}$ and $G^{r^*_{i,:}}$ respectively.

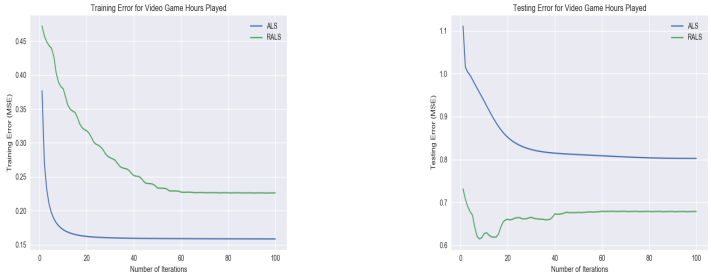
The main idea behind this modification is that it does its best to approximate R with the product of one row/column matrices to put as much of the predictive weight it can on the current pair one row/column matrices. And through manipulation of the process matrix multiplication, it subtracts off predictions as it goes so that it essentially produces a series of pairs of one row/column matrices that should contribute decreasingly less to the approximation of R which can be interpreted as pairs left/right singular vectors being weighted by their corresponding singular values.

3 Application and Results

One of the most popular applications of matrix factorization is to build recommender systems. For recommender systems data comes in the form of users giving a certain item a rating which could be represent with an incomplete matrix. By applying the discussed matrix factorization methods to the incomplete matrix will create predictions that will fill in missing elements in the matrix to recommend user new items that they have not rated. In this report I will be comparing the performance of the regular ALS algorithm with the modified RALS proposed above on real-world and simulated data sets. The real-world data sets include a kaggle data set that list a set of users from the popular video game distribution platform Steam and their hours played for the games they owned and the Movielens 100k benchmark data set from grouplens [2].

3.1 Game Hours Played (Continuous)

The kaggle data set listed users' time played for owned games list time played as hours with 11360 users by 3600 games user-item matrix with about 70000 non-zero hours. One thing of note about this data is that a significant proportion of the users in the data set have playtime on only one game (over 50%). These user were dropped during preprocessing of the data set. Due to wide range of play times demonstrated by users over different games playtime was first converted to minutes and then the $\log_{10}$ of these minutes is taken to replace the play times with an implicit always positive rating. To compare the two methods, ALS and RALS, I will be setting the hyper-parameters of number of latent factors (rank) $k = 20$ and the regularization strength $\lambda = 0.1$ and compare their results on both the training and hold out sets with a training split of 75/25 over 100 iterations.

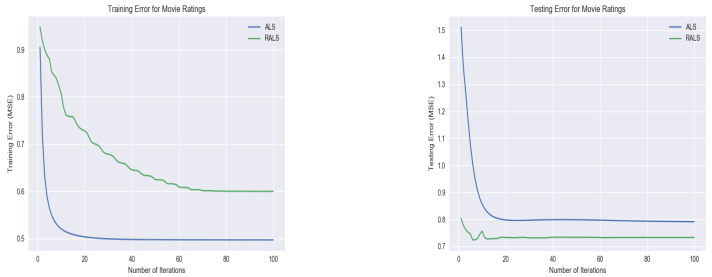


(a) Training Error (b) Testing Error

Figure 1: Training and Testing error for ALS (blue) vs. RALS (green) on the video game playtime data set over 100 iterations

3.2 Movie Ratings (Discrete)

The Movielens data set from grouplens is a popular benchmark data set that with a 100000 ratings for 943 users across 1682 movies. Not much cleaning was done on this data set since its a benchmark set. For the comparison of the two methods the number of latent factors (rank) $k = 20$ and the regularization strength $\lambda = 0.1$ and this was done with 95/5 train split (the split is set so low so that training set doesn't accidentally end up with an empty column or row) over a 100 iterations.



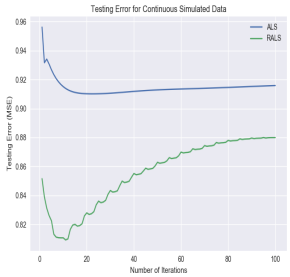
(a) Training Error (b) Testing Error

Figure 2: Training and Testing error for ALS (blue) vs. RALS (green) on the Movielens data set over 100 iterations

3.3 Simulated Data sets

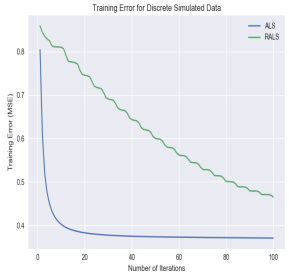
To attempt to simulate data sets similar to the above data sets I assumed that the indices of the sparse observed rating matrix R^* are randomly selected from the complete rating matrix $R \in \mathbb{R}^{m \times n}$ which it self is a set of $m \times n$ samples from a certain distribution. For the continuous data set (log of gameplay minutes) I assumed that the ratings are drawn from a normal distribution with mean 2.48 and standard deviation 0.89. For the discrete data set (movie ratings) I assumed that the rating were sampled from a binomial distribution with $n = 4$ and $p = .632$ and to avoid having ratings of 0 I added one to every rating to have ratings between 1 and 5. Then to create a sparse matrix the indices are selected randomly such that they are no empty columns or rows. For both types of simulated data sets the dimension were set to the same as the Movielens data set (943 users with 1682 items) and a sparsity of about 6%. Then to compare the the two methods across these data sets I set the number of latent factors (rank) $k = 20$ and the regularization strength $\lambda = 0.1$ with a training split of 75/25 over 100 iterations.

(a) Training Error



(b) Testing Error

Figure 3: Training and Testing error for ALS (blue) vs. RALS (green) on simulated continuous ratings over 100 iterations



(a) Training Error

(b) Testing Error

Figure 4: Training and Testing error for ALS (blue) vs. RALS (green) on simulated discrete ratings over 100 iterations

3.4 Discussion of Results

From the figures above one can see that the difference in training error between ALS and RALS becomes smaller and smaller when given enough time and a large enough number of factors k. One can also notice for all data sets (both real and simulated) RALS performed better that ALS in terms of testing error. Although figure 2b does show RALS performing better than ALS, for other 95/5 splits the ALS can end up beating RALS but this can be attributed to low training/testing split that had to be done to preserve non-empty rows and columns in the training set. Also another observation to note is that RALS can also be used to find the best number of factors k to use since each round of RALS can either be set to occur over a set number of iterations or while training one can keep track of when a round of RALS ends and then make predictions on the test with the factor matrices made so far and then pick to stop at the round that results in the lowest error.

4 Conclusion

In conclusion RALS provides an improvement (although small) to the ALS method for matrix factorization and matrix completion. RALS has the potential of being more computationally efficient than ALS especially as the number factors k gets larger since RALS requires finding the inverse of a scalar $n \times m$ times per iteration compared to ALS which requires finding the inverse of a $k \times k$ matrix $n \times m$ times per iterations. Another benefit of RALS is that by looking at the learning curve(testing error versus number of iterations) one can choose k if the number of iterations per round is set before hand or the transition iteration number between rounds are kept track of.

One possible avenue for continuing research and development of RALS and determining its effectiveness is into looking into more sophisticated ways of simulating data sets for this problem. For example; generating data sets that have a predetermined low rank k and seeing how RALS performs on trying to predict entries given a fairly sparse subset. One could also look into possibly imposing the orthogonal constraint from the SVD onto factor matrices' rows/columns. Imposing this constraint while possibly increasing performance could also cause the factor matrices produced to potentially lose some interpretability when it comes to real world data sets.

References

[1] Sean Harnett. "The Netflix Prize: Alternating Least Squares in MPI". In: (2010). DOI: https://pdfs.semanticscholar.org/1b4f/7de1e8d2e1b5b56912894cc61c241ed9b6a3.pdf.

[2] F Maxwell Harper and Joseph A Konstan. "The movielens datasets: History and context". In: *ACM Transactions on Interactive Intelligent Systems (TiiS)* 5.4 (2016), p. 19.

[3] Yunhong Zhou et al. "Large-scale parallel collaborative filtering for the netflix prize". In: *Lecture Notes in Computer Science* 5034 (2008), pp. 337–348.