A Recommendation System for Symmetric Tensor Data

Sol Vitkin and Rishi Sharma

December 20, 2015

Contents

1 Introduction

Imagine you are a schoolteacher or a school district administrator with a wealth of data on how students perform on individual assignments they complete. How well they each do on their assignments tells us much about the students academic ability, but you want to learn more. You want to learn how that student performs or would perform in group activity. We may have that data for some cases, but for the other pairs of students that have not yet worked on a particular assignment, we want to be able to predict their score on that assessment with high accuracy. That example precisely describes the motivation behind this project. For a group of objects, we are looking at and predicting how well an interaction of those objects perform along some metric or over a series of different assessments. We refer to the process of analyzing data in this form as multiway data analysis [8]. Specifically for this problem, we are looking at three modal data, where two of our modes are the same. For up until very recently, much three way data analysis was done simply by flattening the information and stringing out the contents [2]. We however propose a solution to the above mentioned problem, not by flattening, but by instead using tensor factorization and properties of our data's three modal structure to our advantage.

A tensor is nothing but a way to describe n- rank arrays. A 1-rank tensor is a vector and a 2-rank tensor is a matrix. Our project uses a 3-rank tensor. Looking at the Figure 1 below, we see that for a 3-rank tensor, A, we refer to each individual element as $a_{i,j,k}$

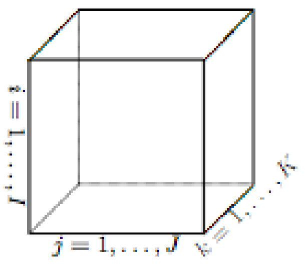
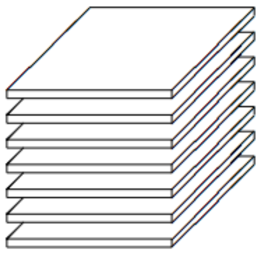


Figure 1: Tensor Representation [2]

As previously mentioned, since we want our data to refer to the interactions of a group of objects with others in its group, our 3-rank tensor is one such that for each cross section, or slice, of that tensor we are representing how the set of objects performs for a given assessment (see Figure 2.)



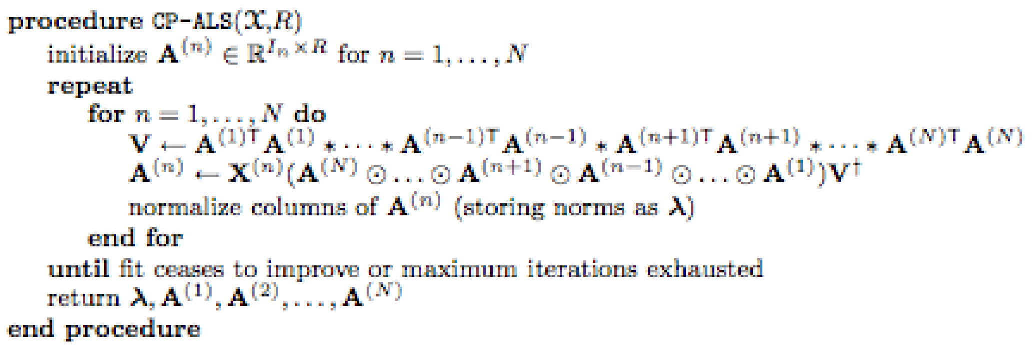
$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} & A_{15} \\ A_{12} & A_{22} & A_{23} & A_{24} & A_{25} \\ A_{13} & A_{23} & A_{33} & A_{34} & A_{35} \\ A_{14} & A_{24} & A_{34} & A_{44} & A_{45} \\ A_{15} & A_{25} & A_{35} & A_{45} & A_{55} \end{bmatrix}$$

Figure 2: a) Slices of a Tensor b) Top Down View of Tensor Slice [2]

Now, for a given assessment k, we notice that how object i performed on that assessment with object j is the same as how object j performed on the assessment with object i. Therefore, for each slice of our tensor we have a symmetric matrix. Also, keep in mind that since we are trying to predict missing values, we expect some level of data sparsity. So, for our data we have a 3-rank $a \times a \times N$ symmetric and sparse tensor.

Many algorithms that use tensor data to make predictions in fields such as computer vision, neuroscience, signal processing etc. use tensor factorization and decomposition to accurately determine the missing data [2]. Hitchcock first theorized tensor decomposition in 1927 [3], and Tucker developed it further for computing, it predicts values by using Hitchcock's observation that a tensor is a sum of component 1-rank tensors. The term factor matrices are used to refer to combination of the vectors from the 1-rank components, and the results are often used to find suitable predictions for missing values. A popular algorithm for dealing with symmetric data, like our data, is the INDSCAL algorithm [2]. The Individual Differences in Scaling (INDSCAL) Algorithm when applied to this problem uses factorization but, keeping in mind symmetry, it assumes that the first two of the three factor matrices are equal. It then typically uses the CP algorithm to compute the factorization. The pseudo-code for the CP algorithm is given by Figure 3 below.

It uses the Alternating Least Squares [4], which iteratively optimizes for a factor matrix, while fixing other factor matrices. Ten Berge et al. [4] shows the assumption that the first two factor matrices are equal does not always result with INDSCAL, and, so, the resulting



```
procedure CP-ALS($\mathcal{X}$,$R$)
    initialize $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R}$ for $n = 1, \dots, N$
    repeat
        for $n = 1, \dots, N$ do
            $\mathbf{V} \leftarrow \mathbf{A}^{(1)\mathsf{T}}\mathbf{A}^{(1)} * \dots * \mathbf{A}^{(n-1)\mathsf{T}}\mathbf{A}^{(n-1)} * \mathbf{A}^{(n+1)\mathsf{T}}\mathbf{A}^{(n+1)} * \dots * \mathbf{A}^{(N)\mathsf{T}}\mathbf{A}^{(N)}$
            $\mathbf{A}^{(n)} \leftarrow \mathbf{X}_{(n)}(\mathbf{A}^{(N)} \odot \dots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \dots \odot \mathbf{A}^{(1)})\mathbf{V}^{\dagger}$
            normalize columns of $\mathbf{A}^{(n)}$ (storing norms as $\boldsymbol{\lambda}$)
        end for
    until fit ceases to improve or maximum iterations exhausted
    return $\boldsymbol{\lambda}, \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \dots, \mathbf{A}^{(N)}$
end procedure
```

Figure 3: Pseudocode for the CP-Algorithm [2]

matrix is not always exactly symmetric. Our paper offers a new solution that uses a set of pattern matrices and weights to predict each tensor slice matrix. We use a coordinate descent algorithm described below to converge on a symmetric solution. Please note that since we could not find clean and usable data in our time frame, all further references to data refer to data that we generated.

2 Problem Formulation

The problem we formulate is to predict missing elements of a 3-tensor. Along the "assessment mode" slices of this tensor will be symmetric student by student matrices. Each entry that is filled in this matrix will have a score for student pairs(along the diagonal, the score for the student working by them self), for a particular assessment. For each of these symmetric slices, we are attempting to predict the missing elements. We will now introduce some basic assumptions we make that allows our algorithm to effectively model the true values. The notation used in these assumptions will stay consistent through the rest of the sections.

Data Assumptions
Essentially we are assuming that each symmetric slice of the tensor can be represented by a linear combination of pattern matrices.

- N number of T_n tensor slices
  - full diagonal: each individual works with themselves
  - symmetric: each member in pair gets same score for an assessment
- I number of P_i symmetric pattern matrices
  - same pattern matrices for each T_n
- Corresponding $\alpha_{i,n}$ scalars for each P_i at each T_n tensor slice
  - α is an I x N matrix whose n^{th} column contains the α_i's for each P_i pattern matrix of the T_n slice.

Our α scalar weight matrix is constrained to be sparse because we do not want to assume equal "importance" of pattern matrices for each tensor slice. In our algorithm and data generation we initialize these $\alpha_{i,n}$'s to be between 0 and 1.

These assumptions for our data allow us to formulate the prediction problem into a convex minimization.

$$\underset{P_1 \ldots P_I, \alpha}{\text{minimize}} \qquad \sum_{n=1}^{N} \frac{1}{2} \|(T_n - \sum_{i=1}^{I} \alpha_{i,n} P_i)_\Omega\|_F^2 + \lambda \|\alpha\|_1$$

subject to:

$$\begin{aligned} P_i^T &= P_i \ \forall i \\ 1 \leq &(P_i)_{k,s} \leq 100 \ \forall i, k, s \end{aligned}$$

Here we are minimizing the frobenius norm of the difference between each tensor slice of data and a linear combination of pattern matrices for the observed elements(our prediction), while also maintaining sparsity for our α matrix with the L-1 norm. In the example of students and assignments, we can think of the pattern matrices as patterns of how students work together. For the constraints, we assume that the scores for the assessments range from 1 to 100, so no element of our pattern matrices is outside of this interval. Further, our pattern matrices must be symmetric to maintain a symmetric linear combination prediction of our symmetric tensor slice. Because scalar multiplication and matrix addition between two symmetric matrices preserves symmetry, this constraint refers to all of our pattern matrices. We split up our minimization into an error term, f($\cdot$), and a regularization term, g($\cdot$), the conventional way:

$$f(\cdot) = \sum_{n=1}^{N} \frac{1}{2} \|(T_n - \sum_{i=1}^{I} \alpha_{i,n} P_i)_\Omega\|_F^2$$

$$g(\cdot) = \lambda \|\alpha\|_1$$

3 Algorithm and Data Generation

For implementation of our algorithm and data generation the NumPy package in Python was primarily used[1].

Algorithm

To minimize our objective function we will use a form of co-ordinate descent, where we minimize each co-ordinate with a gradient descent step. In our formulation, we have two types of coordinates: P_i pattern matrices and $\alpha_{i,n}$ scalars. As mentioned in the introduction, we have an a x a x N tensor. The pseudocode is below:

Initialization:
Initialize I random a x a symmetric random matrices with elements between 1 and 100;
Initialize random α - I x N matrix with elements between 0 and 1;
Initialize η;
Initialize λ;
Initialize M;
m = 0;

Let pattern_list be list of P_i's.
Let alpha_list be list of $\alpha_{i,n}$.

```
while $m < M$ do
    for $P_k$ in pattern_list do
        $\gamma_{P_k} = \frac{1}{\nabla^2_{P_k} f(\cdot)}$;
        update $P_k = Proj_\Theta(P_k - \gamma_{P_k} \nabla_{P_k} f(\cdot))$;
    end
    for $\alpha_{k,s}$ in alpha_list do
        $\gamma_{\alpha_{k,s}} = \frac{1}{\nabla^2_{\alpha_{k,s}} f(\cdot)}$;
        update $\alpha_{k,s} = sgn(\alpha_{k,s} - \nabla_{\alpha_{k,s}} f(\cdot)) \odot \max(0, |\alpha_{k,s} - \nabla_{\alpha_{k,s}} f(\cdot)| - \gamma_{\alpha_{k,s}} \lambda)$ ;
    end
    if $objective_{m-1}$ - $objective_m < \eta$ then
        break
    end
    m = m + 1
end
```

Algorithm 1: Optimization

The algorithm loops through each type of co-ordinate and depending on its type, updates the co-ordinate appropriately, fixing all other co-ordinates. At each co-ordinate a safe step-length of $\frac{1}{\nabla^2 f(\cdot)}$ is used to descend along its gradient. The update step for the pattern matrices takes the previous pattern matrix and subtracts the gradient of the error term of our objective function(in this case a matrix) and uses $Proj_\Theta$ to make sure the elements of the matrix remain between 1 and 100. $Proj_\Theta$ is simply a cutoff function returning 1 if an element of the matrix is below 1 and 100 if the element is above 100. The important aspect of this update step is that it maintains the symmetry of the pattern matrices. The only operations being performed when calculating the gradient and the update step is scalar multiplication and matrix addition and subtraction. Since these operations are performed on already symmetric matrices as initialized, no other operations need to be performed to maintain symmetry in our predictions. The relevant gradient is:

$$\nabla_{P_k} f(\cdot) = \sum_{n=1}^{N} (T_n - \sum_{i=1}^{I} \alpha_{i,n} P_i)_\Omega \, (-\alpha_{k,n})$$

For each $\alpha_{k,s}$ co-ordinate, the regularization term present in our objective function means that the gradient descent step has a closed form solution different than what would normally be expected, as seen in our algorithm[6]. The relevant gradient is:

$$\nabla_{\alpha_{k,s}} f(\cdot) = \sum_{\Omega} (T_s - \sum_{i=1}^{I} \alpha_{i,n} P_i)(-P_k)$$

The derivations of these gradients as well as the second gradients are in the Math Appendix.

The algorithm ends when the difference between the objective function value at iteration *m* and iteration *m-1* is smaller than some η.

Tuning the Parameters: To optimize our regularization term λ, we employed a leave-one out cross validation procedure with our previously described optimization algorithm. The data is split into a training and testing data set. From there, over a range of potential λ's, the optimization algorithm is run on the training set, and the linear combination of resulting pattern matrices is used to predict the elements of the testing set. An RMSE is recorded between the actual elements of the testing set and our prediction, and is compared for different values of λ until the best RMSE is obtained, at which point the optimization algorithm is run on the whole data set with that corresponding best λ. We search for λ by iterating through a list of lambdas. In order to maximize efficiency, we use the facts that the RMSE is convex and that lambda is simply a coefficient which when for a varying lambda yields convex results. Therefore, in a manner similar to a binary search, we start at the middle of the list and look to a value less than it and one greater than it, and choose the one that yields a lower RMSE as our next starting point. We eliminate the values less than the old starting point. We stop the loop when the difference between the current RMSE and the minimum RMSE at that point is less than some small number.

Data Generation

Since as previously mentioned we do not have real world data to test our algorithm on, data was generated according to our model, so we can see how well our algorithm recovers true values. For all tests, we maintained the same dimensions of our tensor: 50 x 50 x 50. To create the tensor slices that made up this tensor, we assumed 15 pattern matrices in our underlying model. We randomly generated a sparse α(with elements between 0 and 1) and created a linear combination of randomly generated pattern matrices and their corresponding $\alpha_{i,n}$ to make each T_n slice. These were stored for comparisons with our predictions later. For the actual data our algorithm used, we randomly and symmetrically removed 75% of the elements of the original slices(except for the diagonal). These are the sparse slices our algorithm then uses for prediction. The underlying pattern matrices are randomly generated under two distributions: a uniform distribution ranging from 1 to 100 and a normal distribution (to simulate grades).

Regardless of how the data was generated, pattern matrices for our algorithm were initialized under the uniform distribution.

4 Results

To see how well our algorithm works, we ran our procedure ten times on the generated data(using the two different types of distributions mentioned earlier) and recorded the average RMSE for these two sets of ten run throughs. The RMSE was taken as the average RMSE of all tensor slices compared to our predicted linear combination of pattern matrices. To put the RMSE's into context, the average range for the generated tensor slices under the uniform distribution is approximately 20 and under the normal distribution, approximately 13.

	Average RMSE
Uniform	4.009
Normal	2.34

Below we present plots of the objective function values per iteration, under the two different distributions. These values were taken from a single run through of our algorithm. The average time for our optimization algorithm(taken over 10 single non-tuned run throughs) with normal data is 94.98 seconds, and with uniform data, 99.95 seconds.

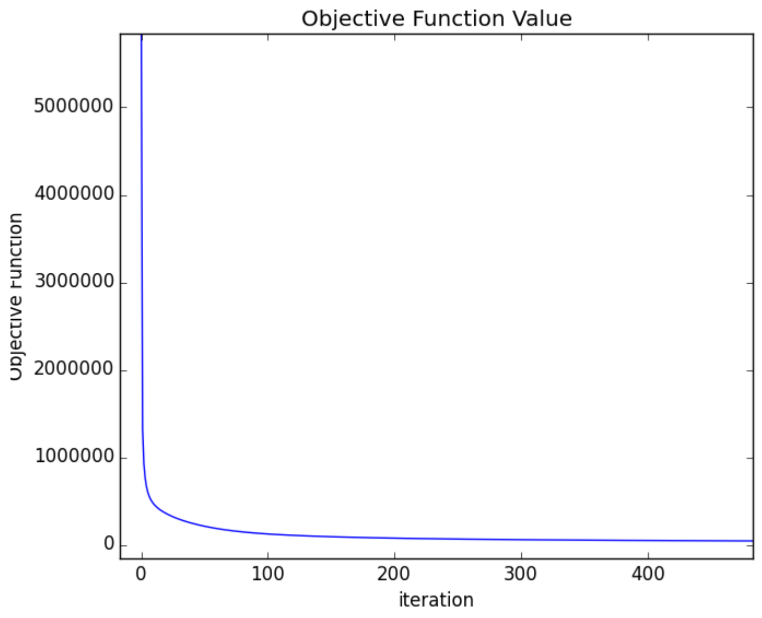


Figure 4: Objective Function Values for Uniformly Generated Data

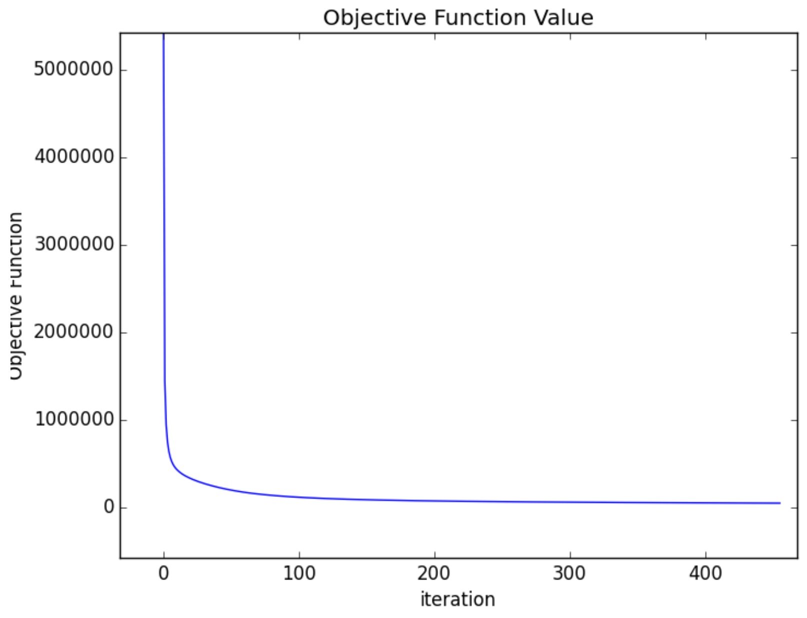


Figure 5: Objective Function Values for Normally Generated Data

5 Conclusion

We believe that the algorithm presented in this paper provides an interesting new perspective and method for predicting values of ”interaction” data - data that has information on how individuals and pairs perform on various assessments, activities, etc. In the future, we hope to find real data that we can apply our procedure to, examine our data assumptions, and find new ways to tune our parameters.

Our algorithm has the potential to provide teachers new ideas for ways to group students on assignments. We also see applications beyond a classroom setting. Our algorithm abstracts for any two identical sets and a set of indexed ”assessments”. Therefore, we hope this spurs further data collection on interactions, as well as further research into predictions on this type of data, creating new opportunities for understanding and motivating interaction.

Acknowledgements
We would like to thank Ji Liu for helping us formulate the problem and helping with writing the algorithm out.

References

[1] Stfan van der Walt, S. Chris Colbert and Gal Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science and Engineering, 13, 22-30 (2011), DOI:10.1109/MCSE.2011.37 (publisher link)

[2] T.G. Kolda and B.W. Bader. Tensor decompositions and applications. SIAM Review, 51(3):(in print), September 2009

[3] F. L. Hitchcock, The expression of a tensor or a polyadic as a sum of products, Journal of Mathematics and Physics, 6 (1927), pp. 164?189.

[4] J. D. Carroll and J. J. Chang, Analysis of individual differences in multidimensional scaling via an N-way generalization of ?Eckart-Young? decomposition, Psychometrika, 35 (1970), pp. 283?319.

[5] J. M. F. Ten Berge, H. A. L. Kiers, and J. de Leeuw, Explicit CANDECOMP/PARAFAC solutions for a contrived 222 array of rank three, Psychometrika, 53 (1988), pp. 579?583

[6] Ji Liu (2015) Gradient Descent. Retrieved from http://www.cs.rochester.edu/ jliu/CSC-576/class-note-7.pdf

[7] Ji Liu (2015) Coordinate Gradient Descent. Retrieved from http://www.cs.rochester.edu/ jliu/CSC-576/class-note-8.pdf

[8] Pieter M. Kroonenberg (2008) Applied Multiway Data Analysis, Hoboken, John Wiley and Sons, Inc.

6 Math Appendix

$$f(\cdot) = \sum_{n=1}^{N} \frac{1}{2} \|(T_n - \sum_{i=1}^{I} \alpha_{i,n} P_i)_\Omega\|_F^2$$

$$\begin{aligned}
f(\cdot) &= \sum_{n=1}^{N} \frac{1}{2} \sqrt{\sum_{j,k\in\Omega} ((T_n)_{j,k} - \sum_{i=1}^{I} (\alpha_{i,n} P_i)_{j,k})^2}^{\,2} \\
&= \sum_{n=1}^{N} \frac{1}{2} \sum_{j,k\in\Omega} ((T_n)_{j,k} - \sum_{i=1}^{I} (\alpha_{i,n} P_i)_{j,k})^2 \\
\nabla_{P_k} f(\cdot) &= \sum_{n=1}^{N} \, (T_n - \sum_{i=1}^{I} \alpha_{i,n} P_i)_\Omega \, (-\alpha_{k,n})
\end{aligned}$$

For the gradient in terms of $\alpha_{k,s}$, the derivative relates to a specific s tensor slice, and some pattern k pattern matrix. Therefore, the summation for the tensor slices disappears when taking the derivative, as any non T_s slice terms go to 0. Using the chain rule in a similar way to the last gradient, the derivative becomes:

$$\nabla_{\alpha_{k,s}} f(\cdot) = \sum_{\Omega} \, (T_s - \sum_{i=1}^{I} \alpha_{i,n} P_i)(-P_k)$$

For the second gradients, the product rule and chain rule is used.

$$\begin{aligned}
\nabla_{P_k} f(\cdot) &= \sum_{n=1}^{N} \, (T_n - \sum_{i=1}^{I} \alpha_{i,n} P_i)_\Omega \, (-\alpha_{k,n}) \\
\nabla_{P_k}^2 f(\cdot) &= \sum_{n=1}^{N} \, (-\alpha_{k,n})(-\alpha_{k,n}) + (T_n - \sum_{i=1}^{I} \alpha_{i,n} P_i)_\Omega (0) \\
&= \sum_{n=1}^{N} \alpha_{k,n}^2
\end{aligned}$$

$$\begin{aligned}
\nabla_{\alpha_{k,s}} f(\cdot) &= \sum_{\Omega} \, (T_s - \sum_{i=1}^{I} \alpha_{i,n} P_i)(-P_k) \\
\nabla_{\alpha_{k,s}}^2 f(\cdot) &= \sum_{\Omega} \, (-P_k)(-P_k) + (T_s - \sum_{i=1}^{I} \alpha_{i,n} P_i)(0) \\
&= \sum_{\Omega} P_k^2
\end{aligned}$$