CS229 PROJECT: FACTOR-ANALYSIS WITH PARTIALLY OBSERVED TRAINING DATA

PAUL CSONKA, BARRETT HEYNEMAN, SALOMON TRUJILLO

1. Motivation

Factor Analysis is typically applied to problems when we have a set of m training examples $x^{(i)} \in \mathbb{R}^n$, where $n \gg m$. We then model the $x^{(i)}$'s as generated by a latent random variable $z^{(i)} \in \mathbb{R}^k$ and noise term $\epsilon \in \mathbb{R}^n$ with a diagonal covariance Ψ, where

$$\begin{aligned} x^{(i)} &= \mu + \Lambda z^{(i)} + \epsilon \\ z &\sim \mathcal{N}(0, I) \\ \epsilon &\sim \mathcal{N}(0, \Psi) \end{aligned} \tag{1}$$

or equivalently

$$x \sim \mathcal{N}(\mu, \Lambda\Lambda^T + \Psi) \tag{2}$$

Consider a similar situation in which, instead of the full vector $x^{(i)}$, we only observe some part of $x^{(i)}$; a vector $v^{(i)} \in \mathbb{R}^{n^{(i)}}$, $n^{(i)} \leq n$, such that

$$v^{(i)} = {J^{(i)}}^T x^{(i)} \tag{3}$$

$J^{(i)} \in \mathbb{R}^{n \times n^{(i)}}$ is a full rank matrix, with orthonormal columns. If we let $r^{(i)} \in \mathbb{R}^{n^{(i)}}$ consist of the indices of $x^{(i)}$ which are observed, i.e. present in $v^{(i)}$, then

$$J^{(i)} = \begin{bmatrix} \mathbf{e}_{r_1^{(i)}} & \mathbf{e}_{r_2^{(i)}} & \cdots \end{bmatrix} \tag{4}$$

We can then express the distribution of $v^{(i)}$ as

$$v^{(i)} \sim \mathcal{N}\left({J^{(i)}}^T \mu, {J^{(i)}}^T \left(\Lambda\Lambda^T + \Psi\right) J^{(i)}\right) \tag{5}$$

If we define the following variables

$$\begin{aligned} \mu^{(i)} &= {J^{(i)}}^T \mu \\ \Lambda^{(i)} &= {J^{(i)}}^T \Lambda \\ \Psi^{(i)} &= {J^{(i)}}^T \Psi J^{(i)} \end{aligned} \tag{6}$$

then we can express the joint distribution of $z^{(i)}$ and $v^{(i)}$ as

$$\begin{bmatrix} z \\ v^{(i)} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \vec{0} \\ \mu^{(i)} \end{bmatrix}, \begin{bmatrix} I & {\Lambda^{(i)}}^T \\ \Lambda^{(i)} & \Lambda^{(i)}{\Lambda^{(i)}}^T + \Psi^{(i)} \end{bmatrix} \right) \tag{7}$$

2. The E-step

To apply the E-step of the EM algorithm to our modified factor analysis problem we need the conditional distrubition $z \mid v^{(i)}; \mu^{(i)}, \Lambda^{(i)}, \Psi^{(i)} \sim \mathcal{N}(\mu_{z^{(i)}|v^{(i)}}, \Sigma_{z^{(i)}|v^{(i)}})$ where

$$\begin{aligned} \mu_{z^{(i)}|v^{(i)}} &= {\Lambda^{(i)}}^T \left(\Lambda^{(i)}{\Lambda^{(i)}}^T + \Psi^{(i)}\right)^{-1} \left(v^{(i)} - \mu^{(i)}\right) \\ \Sigma_{z^{(i)}|v^{(i)}} &= I - {\Lambda^{(i)}}^T \left(\Lambda^{(i)}{\Lambda^{(i)}}^T + \Psi^{(i)}\right)^{-1} \Lambda^{(i)} \end{aligned}$$

Therefore, $Q_i(z^{(i)})$ is simply the *pdf* of this distribution.

3. The M-step

The M-step is not as straight forward as the E-step, and requires results found in Appendix A. After removing terms that don't depend on the parameters and replacing the integral over $z^{(i)}$ with an expectation, the algorithm must maximize the log likelihood of the data, which is

$$\begin{aligned} &\sum_{i=1}^{m} \mathrm{E}_{z^{(i)}\sim Q_i}\left[\log p\left(v^{(i)} \mid z^{(i)}; \mu^{(i)}, \Lambda^{(i)}, \Psi^{(i)}\right)\right] \\ &= \sum_{i=1}^{m} \mathrm{E}_{z^{(i)}\sim Q_i}\left[-\frac{1}{2}\log|\Psi^{(i)}| - \frac{n}{2}\log(2\pi) \right. \\ &\qquad \left. -\frac{1}{2}\left(v^{(i)} - \mu^{(i)} - \Lambda^{(i)}z^{(i)}\right)^T {\Psi^{(i)}}^{-1}\left(v^{(i)} - \mu^{(i)} - \Lambda^{(i)}z^{(i)}\right)\right] \end{aligned} \tag{8}$$

Here $\mathrm{E}_{z^{(i)}\sim Q_i}$ indicates the expectation is with respect to $z^{(i)}$ drawn from the distribution Q_i. Since there is no ambiguity, we will drop the "$z^{(i)} \sim Q_i$" subscript.

3.1. M-Step for Λ.

First, the log-likelihood is maximized w.r.t. Λ. We must replace $\Lambda^{(i)}$ and $\mu^{(i)}$ with ${J^{(i)}}^T\Lambda$ and ${J^{(i)}}^T\mu$ respectively and then take the gradient w.r.t. Λ. Dropping terms with no Λ dependence we get

$$\begin{aligned} &\nabla_\Lambda \sum_{i=1}^{m} -\mathrm{E}\left[\frac{1}{2}\left(v^{(i)} - \mu^{(i)} - \Lambda^{(i)}z^{(i)}\right)^T {\Psi^{(i)}}^{-1}\left(v^{(i)} - \mu^{(i)} - \Lambda^{(i)}z^{(i)}\right)\right] \\ &= \sum_{i=1}^{m} \nabla_\Lambda \mathrm{E}\left[-\operatorname{tr}\frac{1}{2} {z^{(i)}}^T \Lambda^T J^{(i)} {\Psi^{(i)}}^{-1} {J^{(i)}}^T \Lambda z^{(i)} \right. \\ &\qquad \left. + {z^{(i)}}^T \Lambda^T J^{(i)} {\Psi^{(i)}}^{-1}\left(v^{(i)} - {J^{(i)}}^T\mu\right)\right] \\ &= \sum_{i=1}^{m} \mathrm{E}\left[-J^{(i)}{\Psi^{(i)}}^{-1}{J^{(i)}}^T\Lambda z^{(i)}{z^{(i)}}^T + J^{(i)}{\Psi^{(i)}}^{-1}\left(v^{(i)} - {J^{(i)}}^T\mu\right){z^{(i)}}^T\right] \end{aligned}$$

Substituting the definition of ${\Psi^{(i)}}^{-1}$ from 14 yields

$$= \sum_{i=1}^{m} \mathrm{E}\Big[- J^{(i)} {J^{(i)}}^T \Psi^{-1} J^{(i)} {J^{(i)}}^T \Lambda z^{(i)} {z^{(i)}}^T + \operatorname{tr} J^{(i)} {J^{(i)}}^T \Psi^{-1} J^{(i)} \left(v^{(i)} - {J^{(i)}}^T \mu \right) {z^{(i)}}^T \Big]$$

$$= \sum_{i=1}^{m} \mathrm{E}\left[-\Delta^{(i)} \Psi^{-1} \Delta^{(i)} \Lambda z^{(i)} {z^{(i)}}^T + \Delta^{(i)} \Psi^{-1} \left(J^{(i)} v^{(i)} - \Delta^{(i)} \mu \right) {z^{(i)}}^T \right]$$

Using the fact that diagonal matrices commute, and the result from 17 we get

$$= \sum_{i=1}^{m} \mathrm{E}\left[-\Psi^{-1} \Delta^{(i)} \Lambda z^{(i)} {z^{(i)}}^T + \Psi^{-1} \Delta^{(i)} \left(J^{(i)} v^{(i)} - \Delta^{(i)} \mu \right) {z^{(i)}}^T \right]$$

Because $\Delta^{(i)}$ is not necessarily full rank, we can not at this point set the equation equal to zero and solve for Λ. However, if we set the equation equal to zero and multiply by ${\mathbf{e}_j}^T$, for $j = 1..n$, we are left with n independent equations,

$$\sum_{i=1}^{m} \mathrm{E}\left[-{\mathbf{e}_j}^T \Psi^{-1} \Delta^{(i)} \Lambda z^{(i)} {z^{(i)}}^T + {\mathbf{e}_j}^T \Psi^{-1} \Delta^{(i)} \left(J^{(i)} v^{(i)} - \Delta^{(i)} \mu \right) {z^{(i)}}^T \right] = 0$$

Since Ψ^{-1} and $\Delta^{(i)}$ are both diagonal, with j^{th} entries $\frac{1}{\psi_j}$ and δ_{ij} respectively, then ${\mathbf{e}_j}^T \Psi^{-1} \Delta^{(i)} = \frac{\delta_{ij}}{\psi_j} {\mathbf{e}_j}^T$. Also, using 17 gives us

$$= \sum_{i=1}^{m} \mathrm{E}\left[-\frac{\delta ij}{\psi_j} {\mathbf{e}_j}^T \Lambda z^{(i)} z(i)^T + \frac{\delta ij}{\psi_j} \left({\mathbf{e}_j}^T J^{(i)} v^{(i)} - {\mathbf{e}_j}^T \mu \right) {z^{(i)}}^T \right]$$

Since $\delta_{ij} = 1$ iff $i \in S_j$, we can instead sum just over the set S_j and drop the δ_{ij}'s.

$$= \sum_{i \in S_j} \mathrm{E}\left[-\frac{1}{\psi_j} {\mathbf{e}_j}^T \Lambda z^{(i)} {z^{(i)}}^T + \frac{1}{\psi_j} \left({\mathbf{e}_j}^T J^{(i)} v^{(i)} - {\mathbf{e}_j}^T \mu \right) {z^{(i)}}^T \right]$$

${\mathbf{e}_j}^T \Lambda$ selects ${\lambda_j}^T$, the j^{th} row of Λ, allowing us to independently solve for the n rows of Λ. Additionally, ${\mathbf{e}_j}^T J^{(i)} v^{(i)} = x_j^{(i)}$ and ${\mathbf{e}_j}^T \mu = \mu_j$, which allows us to further simplify the equation as

$$\sum_{i \in S_j} \mathrm{E}\left[\frac{1}{\psi_j} {\lambda_j}^T z^{(i)} {z^{(i)}}^T \right] = \sum_{i \in S_j} \mathrm{E}\left[\frac{1}{\psi_j} \left(x_j^{(i)} - \mu_j \right) {z^{(i)}}^T \right]$$

$$\sum_{i \in S_j} \mathrm{E}\left[{\lambda_j}^T z^{(i)} {z^{(i)}}^T \right] = \sum_{i \in S_j} \mathrm{E}\left[\left(x_j^{(i)} - \mu_j \right) {z^{(i)}}^T \right]$$

$$ {\lambda_j}^T \sum_{i \in S_j} \mathrm{E}\left[z^{(i)} {z^{(i)}}^T \right] = \sum_{i \in S_j} \left(x_j^{(i)} - \mu_j \right) \mathrm{E}\left[{z^{(i)}}^T \right]$$

$$ {\lambda_j}^T = \left(\sum_{i \in S_j} \left(x_j^{(i)} - \mu_j \right) \mathrm{E}_{z^{(i)} \sim Q_i}\left[{z^{(i)}}^T \right] \right) \left(\sum_{i \in S_j} \mathrm{E}_{z^{(i)} \sim Q_i}\left[z^{(i)} {z^{(i)}}^T \right] \right)^{-1}$$

And after applying the rules for the expectation of ${z^{(i)}}^T$ and $z^{(i)}{z^{(i)}}^T$:

$$(9) \qquad {\lambda_j}^T = \left(\sum_{i \in S_j} \left(x_j^{(i)} - \mu_j \right) \mu_{z^{(i)}|x^{(i)}}^T \right) \left(\sum_{i \in S_j} \mu_{z^{(i)}|x^{(i)}} \mu_{z^{(i)}|x^{(i)}}^T + \Sigma_{z^{(i)}|x^{(i)}} \right)^{-1}$$

Since each row ${\lambda_j}^T$ deals with a different set S_j, each must be calculated seperately and concatenated to form the full Λ.

4. M-Step for μ and Ψ

Similarly, the log-likelihood must be maximized w.r.t. μ and Ψ. These two derivations are very similar, and are therefore worked out in Appendix B. Only the resulting equations are shown here:

$$(10) \qquad \mu_j = \frac{1}{m^{(j)}} \sum_{i \in S_j} \left(x_j^{(i)} - {\lambda_j}^T \mu_{z^{(i)}|x^{(i)}} \right)$$

$$(11) \qquad \psi_j = \frac{1}{m} \sum_{i \in S_j} \left(\left(x_j^{(i)} - \mu_j - {\lambda_j}^T \mu_{z^{(i)}|v^{(i)}} \right)^2 + {\lambda_j}^T \Sigma_{z^{(i)}|v^{(i)}} \lambda_j \right)$$

5. Algorithm Results

The algorithm was implemented in Matlab; both Pseudo-Code of this implementation and more in depth results of tests and NetFlix iterations are presented in Appendix C and D, respectively.

Several extensive tests were conducted over a wide range of parameter values, in order to validate the algorithm. The parameters include Σ, m/n, k, γ (fraction of remaining training data), and Relative Noise. The bulk of our empirical testing was done on three Windows machines: one to run full-sized Netflix predictions, and two desktops for parameter variation. The approximate times for one test were 17 hours for a single NetFlix iteration, and 5-10 hours for the two desktops running one complete multiple-parameter test. Results are shown in Appendix D.

Our Netflix results indicate that the algorithm does in fact converge, with returned accuracies reported by NetFlix to improve from r^2 =1.29 to 1.15, from the 1^{st} to the 4^{th} iteration. However, the accuracy appears to be approaching an asymptotic limit, leading us to believe that there may be several improvements required before we could achieve better results. Specifically, next we should remove outlier users, and take the penalty of predicting for them inaccurately, with a Λ matrix that does not include those users. However, presumably with the improvement in accuracy for the bulk of the users, our overall error decreases. We have the cutoffs determined, but have not implemented this portion thus far. We have also seen that for a given randomly generated data and user-preferences set, specific users consistently contribute to high condition numbers K for the inverted matrix $\Lambda\Lambda^T + \Psi$; removing these users in the manner described should improve our predictions. This has not been tested yet.

The two desktops ran a battery of tests to find optimal performance based on varying parameters. First, we have confirmed a few obvious properties of the algorithm, in order to verify that we are running correctly: for example, the r^2 value

remains relatively high until about 30 percent of the training data is retained, whereupon most m/n ratios converge to the same final prediction accuracy. We tested m/n ranging from 1:1 to 200:1. The accuracy of predictions improved for the sets with more data, as the ratio increased. After about 20:1, though improvement is observed, it is not significant. For the NetFlix challenge, this actual ratio of the provided data is about 30:1, this is inside our found optimal region–at least, for the algorithm we presented, and for the data generated. However, with approximately only 1 percent of the training data present in NetFlix, we can safely confirm that this is a hard problem...

As the number of features, k, increases, computation becomes much more expensive, going at least as k^2. Based on observing the iteration time for a single convergence, for a representative range of parameters, we've found that for the majority of m/n, roughly $k \leq n/2$ gives the fastest results, with not much significant improvement for very small k; the overhead of the rest of the code might be dominating here. For roughly $k \geq n$, the computation becomes many times more lengthy, especially for large m/n ratios (please see Fig.5).

It appears that when taking into account r^2 accuracy and trying to limit computation time, the optimal values (assuming parameters other than m/n fixed) are roughly $n/2 \leq k \leq n$. If this holds for the NetFlix data, then the maximum $k = 40$ we have used for the full data set is too small. k's larger than 55 crash our computers, and so aside from a single iteration becoming impractically long, memory limits prohibit testing this result on our Challenge submissions. Note that with a larger fraction of data left out during training–more than 80 percent–and with high m/n ratios, for small k ($k \leq 20$) the iteration times are found to be roughly 2-3 times as high as for runs with $k > 20$. Essentially, this expresses what we've noted previously: when there is lots of missing data ($m \gg n$), having a small k means that it is very difficult to accurately reproduce the underlying structure and correlation matrix, leading to more cycles to convergence inside each iteration. Hence, the iteration times increase. However, once we have sufficient k, roughly $k \geq n/2$ as shown previously, data reconstruction becomes easier, and iteration times drop several fold despite the larger k.

With these empirical results, we have found some bounds that improve the performance of this specific algorithm. The cumulative results have not been added to our algorithm yet, and remains for future work.

6. Conclusions

The E-M algorithm, as defined by these updates, runs until convergence as indicated by either very small changes in the Frobinius norms of consecutive Λ, Ψ, and μ, or r^2 prediction error. Important observations are

- Optimal performance is reached when at least 30% of the data is observed.
- Training outliers noticably effect prediction accuracy.
- Higher m/n ratios improve prediction accuracy (all other parameters fixed)
- Larger k improves prediction accuracy (all other parameters fixed)
- Computation time becomes prohibitively long for both large m/n ratio and large k, implying a criteria for optimizing the choice of k.

In summary, we believe the developed and tested algorithm is sound, and a useful method for attacking the NetFlix challenge.

Appendix A. Useful Definitions and Relations

This appendix defines expressions and relations which are used in the derivation of the E and M steps of the modified Factor Analysis algorithm.

A.1. **Properties of $J^{(i)}$.** Because $J^{(i)}$ is full rank and it's columns are basis vectors in $\mathbb{R}^n$, we can define a diagonal matrix $\Delta^{(i)} \in \mathbb{R}^{n\times n}$ such that

$$\begin{aligned} J^{(i)}{J^{(i)}}^T &= \Delta^{(i)} \\ {J^{(i)}}^T J^{(i)} &= \mathbf{I}_{n^{(i)}} \end{aligned} \tag{12}$$

From these two definitions we can see that

$$\begin{aligned} J^{(i)} &= J^{(i)}{J^{(i)}}^T J^{(i)} \\ &= \Delta^{(i)} J^{(i)} \end{aligned} \tag{13}$$

A.2. **Inverse of $\Psi^{(i)}$.** Given the definitions of $J^{(i)}$ and $\Psi^{(i)}$, we need to compute ${\Psi^{(i)}}^{-1}$. Since Ψ and $\Psi^{(i)}$ are both diagonal, we expect that

$${\Psi^{(i)}}^{-1} = {J^{(i)}}^T \Psi^{-1} J^{(i)} \tag{14}$$

To prove that this is the case, compute $\Psi^{(i)}{\Psi^{(i)}}^{-1}$ and check the result. Recall that Ψ, Ψ^{-1}, and $\Delta^{(i)}$ are diagonal, and that diagonal matrices commute.

$$\begin{aligned} \Psi^{(i)}{\Psi^{(i)}}^{-1} &= ({J^{(i)}}^T \Psi J^{(i)})({J^{(i)}}^T \Psi^{-1} J^{(i)}) \\ &= {J^{(i)}}^T \Psi \Delta^{(i)} \Psi^{-1} J^{(i)} \\ &= {J^{(i)}}^T \Delta^{(i)} \Psi \Psi^{-1} J^{(i)} \\ &= {J^{(i)}}^T \Delta^{(i)} J^{(i)} \\ &= {J^{(i)}}^T J^{(i)} {J^{(i)}}^T J^{(i)} \\ &= I \end{aligned}$$

Clearly, because of the commutative property of diagonal matrices, the same result holds for ${\Psi^{(i)}}^{-1}\Psi^{(i)}$

A.3. **Relation between $\Delta^{(i)}$, $J^{(i)}$, δ_{ij}, and the set S_j.** Consider what the columns of $J^{(i)}$ mean. If $\mathbf{e}_j$ is a column of $J^{(i)}$ then $x_j^{(i)}$ is observed, i.e. is present in $v^{(i)}$. Therefore, $\Delta^{(i)}$ has another interpretation.

$$\Delta^{(i)}_{j\ell} = \begin{cases} \delta_{ij} & \text{if } j = \ell \\ 0 & \text{if } j \neq \ell \end{cases} \tag{15}$$

where

$$\delta_{ij} = \begin{cases} 1 & \text{if } x_j^{(i)} \text{ is observed} \\ 0 & \text{if } x_j^{(i)} \text{ is unobserved} \end{cases} \tag{16}$$

From this definition one can see that

$$\Delta^{(i)^2} = \Delta^{(i)} \tag{17}$$

We can then define a set S_j equivalently as the set of all training example indices, i, such that

- $x_j^{(i)}$ is observed
- $\mathbf{e}_j$ is a column of $J^{(i)}$
- $\delta_{ij} = 1$
- $\Delta_{jj}^{(i)} = 1$

Finally, we can define the number of examples for which $x_j^{(i)}$ was observed as

$$m^{(j)} = |S_j| \tag{18}$$

Appendix B. M-Step for μ and Ψ

The derivations for μ and Ψ are presented in this section; only the resulting equations were shown in the paper body.

B.1. **M-Step for μ.** The log-likelihood must be maximized w.r.t. μ. First replace $\mu^{(i)}$ and $\Lambda^{(i)}$ with ${J^{(i)}}^T\mu$ and ${J^{(i)}}^T\Lambda$, and then take the derivative w.r.t. μ. Dropping terms with no μ dependance we have

$$\begin{aligned}&\frac{\partial}{\partial\mu}\sum_{i=1}^{m}\mathrm{E}\left[-\frac{1}{2}\left(v^{(i)}-{J^{(i)}}^T\mu-{J^{(i)}}^T\Lambda z^{(i)}\right)^T{\Psi^{(i)}}^{-1}\left(v^{(i)}-{J^{(i)}}^T\mu-{J^{(i)}}^T\Lambda z^{(i)}\right)\right]\\&=\sum_{i=1}^{m}\mathrm{E}\left[-J^{(i)}{\Psi^{(i)}}^{-1}{J^{(i)}}^T\mu+J^{(i)}{\Psi^{(i)}}^{-1}\left(v^{(i)}-{J^{(i)}}^T\Lambda z^{(i)}\right)\right]\end{aligned}$$

Using the definition of ${\Psi^{(i)}}^{-1}$ and $\Delta^{(i)}$ from 14 and 15 respectively we have

$$\begin{aligned}&=\sum_{i=1}^{m}\mathrm{E}\left[-\Delta^{(i)}\Psi^{-1}\Delta^{(i)}\mu+\Delta^{(i)}\Psi^{-1}\left(J^{(i)}v^{(i)}-\Delta^{(i)}\Lambda z^{(i)}\right)\right]\\&=\sum_{i=1}^{m}\mathrm{E}\left[-\Psi^{-1}\Delta^{(i)}\mu+\Psi^{-1}\Delta^{(i)}\left(J^{(i)}v^{(i)}-\Lambda z^{(i)}\right)\right]\end{aligned}$$

To maximize the w.r.t. μ we set the equation equal to zero. Once again, the equation is simplified if we isolate the elements of μ, one at a time, by multiplying both sides by ${\mathbf{e}_j}^T$, and then deal with the resulting n independent equations. Following similar steps as in the Λ derivation we are left with

$$\sum_{i\in S_j}\mathrm{E}\left[\mu_j\right]=\sum_{i\in S_j}\left(x_j^{(i)}-{\lambda_j}^T\mathrm{E}\left[z^{(i)}\right]\right)$$

Since μ_j does not depend on the summation over S_j and is constant, we can replace the left hand side with $m^{(j)}\mu_j$. On the right side we substitute the definition for $\mathrm{E}\left[z^{(i)}\right]$. The update for μ_j becomes

$$\mu_j=\frac{1}{m^{(j)}}\sum_{i\in S_j}\left(x_j^{(i)}-{\lambda_j}^T\mu_{z^{(i)}|x^{(i)}}\right)\tag{19}$$

B.2. **M-Step for Ψ.** To estimate Ψ we maximize 8 with respect to Ψ. Expand 8 by replacing $\mu^{(i)}$, $\Lambda^{(i)}$, and $\Psi^{(i)}$ with ${J^{(i)}}^T\mu$, ${J^{(i)}}^T\Lambda$, and ${J^{(i)}}^T\Psi$ respectively. Then drop terms with no Ψ dependence, take the gradient w.r.t. Ψ, and distribute $J^{(i)}$ to get

$$\nabla_\Psi \sum_{i=1}^{m} \mathrm{E}\Big[-\frac{1}{2}\log|{J^{(i)}}^T \Psi J^{(i)}| \\ -\frac{1}{2}\Big(v^{(i)} - {J^{(i)}}^T\mu - {J^{(i)}}^T\Lambda z^{(i)}\Big)^T {J^{(i)}}^T \Psi^{-1} J^{(i)} \Big(v^{(i)} - {J^{(i)}}^T\mu - {J^{(i)}}^T\Lambda z^{(i)}\Big)\Big] \\ = \sum_{i=1}^{m} \mathrm{E}\left[\Psi^{-1} - \Psi^{-1}\left(J^{(i)}v^{(i)} - \Delta^{(i)}\mu - \Delta^{(i)}\Lambda z^{(i)}\right)\left(J^{(i)}v^{(i)} - \Delta^{(i)}\mu - \Delta^{(i)}\Lambda z^{(i)}\right)^T \Psi^{-1}\right]$$

If we set this equal to 0 and both left and right multiply by Ψ (which is invertible) we find that the equation is maximized by

$$\Psi = \frac{1}{m}\sum_{i=1}^{m} \mathrm{E}\left[\left(J^{(i)}v^{(i)} - \Delta^{(i)}\mu - \Delta^{(i)}\Lambda z^{(i)}\right)\left(J^{(i)}v^{(i)} - \Delta^{(i)}\mu - \Delta^{(i)}\Lambda z^{(i)}\right)^T\right]$$

At this point we use equation 13 to replace the last remaining $J^{(i)}$

$$\Psi = \frac{1}{m}\sum_{i=1}^{m} \mathrm{E}\left[\Delta^{(i)}\left(J^{(i)}v^{(i)} - \mu - \Lambda z^{(i)}\right)\left(J^{(i)}v^{(i)} - \mu - \Lambda z^{(i)}\right)^T {\Delta^{(i)}}^T\right]$$

However, we have restricted Ψ to be diagonal, so let us select only the diagonal elements of both sides of the equation by left and right multiplying by ${\mathbf{e}_j}^T$ and $\mathbf{e}_j$ respectively. Following the same steps as used in the Λ and μ derivations (recall that ${\mathbf{e}_j}^T J^{(i)} v^{(i)} = x_j^{(i)}$ and ${\mathbf{e}_j}^T\mu = \mu_j$) we see that

$$\psi_j = \frac{1}{m}\sum_{i=1}^{m} \mathrm{E}\left[{\mathbf{e}_j}^T\Delta^{(i)}\left(J^{(i)}v^{(i)} - \mu - \Lambda z^{(i)}\right)\left(J^{(i)}v^{(i)} - \mu - \Lambda z^{(i)}\right)^T {\Delta^{(i)}}^T\mathbf{e}_j\right] \\ = \frac{1}{m}\sum_{i\in S_j}\left(x_j^{(i)} - \mu_j\right)^2 + 2\left(x_j^{(i)} - \mu_j\right){\lambda_j}^T \mathrm{E}\left[z^{(i)}\right] + {\lambda_j}^T \mathrm{E}\left[z^{(i)}{z^{(i)}}^T\right]\lambda_j$$

Plugging in for the definitions of the expectation of $z^{(i)}$ and $z^{(i)}{z^{(i)}}^T$ and simplifying yields

$$\psi_j = \frac{1}{m}\sum_{i\in S_j}\left(\left(x_j^{(i)} - \mu_j - {\lambda_j}^T\mu_{z^{(i)}|v^{(i)}}\right)^2 + {\lambda_j}^T\Sigma_{z^{(i)}|v^{(i)}}\lambda_j\right) \tag{20}$$

Appendix C. Implementation Details

The application which motivated development of this algorithm was the Netflix challenge; a challenge to predict users' ratings of unseen movies based on all known rating data. The Netflix data consists of sparse ratings for 17,000 movies by 500,000 users. At first we considered a training example to be the vector of all users' ratings for a given movie, giving $n = 500,000$ and $m = 17,000$. Even considering the sparsity of the data, the square matrix $(\Lambda^{(i)}{\Lambda^{(i)}}^T + \Psi^{(i)})$ would have been, at worst, in $\mathbb{R}^{250,000 \times 250,000}$. Having to repeatedly invert those matrices would have been computationally prohibitive, severely limiting the number of test iterations we would have been able to run.

For that reason we applied the algorithm to the "transpose" problem; i.e. assuming a training example is a vector of a particular user's ratings for all movies. In this format we have many more training examples than the dimension of each training example, which is contrary to the typical motivation for applying factor analysis.

However, we feel that the algorithm is still applicable because the underlying model still holds. Specifically, we feel it is reasonable to assume that users can be decomposed into a weighted sum of k *eigen-users* and that the rating for a given movie can be generated as a linear function of a particular user's *eigen-user* components.

C.1. **MATLAB.** MATLAB was chosen because it is very good at evaluating an expression of the form $A^{-1}B$, which is the primary bottleneck of the algorithm. The algorithm can be parallized across various computers, but to do so, they would need to share intermmediate values, which can amount to approximately 1.5 gigabytes for the Netflix test data. While this is feasible on today's computers, it's not practical to implement on shared resources.

In order to reduce memory contraints, Ψ and J were not stored as matricies, but rather as vectors. Since Ψ is diagonal, it can be stored as vector of its diagonal entires. J is also stored as a vector of indicies that map the element of x to the elements of $J^T x$. In MATLAB, this is accomplished by executing the code `x(J)` for vectors or `X(J,:)` for matricies.

In order to solve $A^{-1}B$, we used the fact that $\Lambda\Lambda^T + \Psi$ is positive definite, and therefore a more efficent algorithm can be applied to solving it. MATLAB implements a processor optimizied BLAS software package, so it stands to be faster than any code we write. MATLAB uses the command `linsolve` to calculate $A^{-1}B$. `linsolve` will take additional flags that indicate that A is postivite definite. This method was the fastest of several tested.

The following psuedo-code was used to update the parameters of the algorithm

```
Allocate memory for variables row, mat, inv_col and inv_submat

Allocate and initialize to zero variables for
  NewLambda, NewPsi, NewMu and m_j

for i = {1 to m}

  Load J and x from disk for user i
```

```
    if dimension(x) > max_dimension
      choose max_dimension random ratings from x
      reform J to match x
    end if

  Solve the E-step for this user

    [inv_col inv_submat] =
      (J'(Lambda * Lambda' + Psi)J)^-1 * [(x - J'mu) J' Lambda]

    mu_zi_xi = Lambda'J * inv_col
    Sigma_zi_xi = I - J'Lambda * inv_submat

  Apply this user's contribution to the M-step

    mu_term = x - J'Lambda * mu_zi_xi
    mat_term = (mu_zi_xi * mu_zi_xi') + Sigma_zi_xi

    for j = {set of movies user i rated}

      row_term = ( x(j) - J'mu(j) ) * mu_zi_xi

      row(j) = row(j) + row_term
      mat(j) = mat(j) + mat_term

      NewPsi(j) = NewPsi(j) +
        J'Lambda(j) * Sigma_zi_xi * Lambda(j)'J +
        (mu_term - mu(j))^2
      NewMu(j) = NewMu(j) + mu_term

      m_j(j) = m_j(j) + 1

    end for

end for

for j = {1 to n}

  NewLambda(j) = row(j) * mat(j)^-1
  NewMu(j) = NewMu(j) / m_j(j)

end for
```

The algorithm took ten hours for a single iteration on the Netflix data. This was acceptable since we want to submit the prediction of each iteration to Netflix, and their system only allows one predicition every twenty-four hours.

C.2. **Extensions of the algorithm.** It would take a quarter tetrabyte to store the largest $(\Lambda^{(i)}\Lambda^{(i)^T} + \Psi^{(i)})$ matrix in double-precision, even while taking advantage of symmetry. However, for iterative linear solving algorithms, such as the conjugate gradient method, the whole matrix would not need to be stored. The algorithm would calculate the residual $r = Ax - b$ and use r to update the value of x. Ax can be calculated in MATLAB without ever storing its explicit form: `Lambda * (Lambda' * x) + diag(Psi .* x)`. However, as it turns out, as $x's$ grow to the 250,000 element size, round-off error becomes a problem and the algorithm fails to converge.

Appendix D. Results

The following section presents more detailed results supporting claims made in the conclusion of the main paper.

D.1. **Repeatability and Effects of Noise.** A data set was generated according to the underlying model for which the algorithm was designed, with parameters $m = n = 100$, $k = 20$. The rows of Λ, $\lambda_j \in \mathbb{R}^k$, and the latent variables $z^{(i)} \in \mathbb{R}^k$ were independently drawn from a gaussian, $\mathcal{N}(0, I)$. The noise terms, $\epsilon \in \mathbb{R}^n$, were also drawn from a gaussian, $\mathcal{N}(0, \sigma^2 I)$. The algorithm was run to convergence 5 times for varying σ^2 values, fraction training data observed, and the assumed value of k. The resulting prediction errors used to generate average and sample variance data.

Prediction accuracy drastically increases around 10% observed data, and again around 50% observed data. Prediction with less than 10% is virtually impossible, being not much better than randomly guessing. By comparing Figs. D.1 and D.1 we see that, as would be expected, when the noise used to generate the data is very small, prediction is much more accurate.

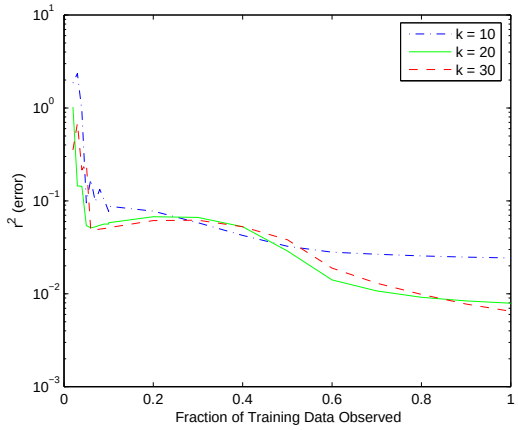

Figure 1. Prediction error for moderate noise; $\sigma^2 = 1$

Figs. D.1 and D.1 indicate that the algorithm is sufficiently repeatable for more than 10% observed data, while suggesting that final prediction is very sensitive to initial value of Λ, Ψ, and μ for any less.

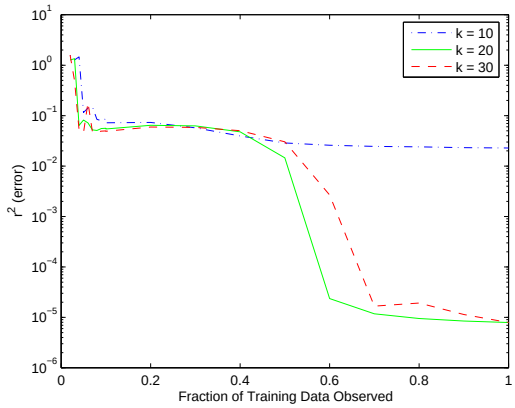

FIGURE 2. Prediction error for moderate noise; $\sigma^2 = 0.001$

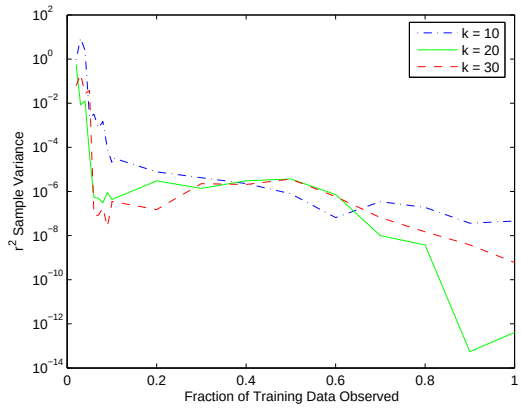

FIGURE 3. Sample variance of prediction error for moderate noise; $\sigma^2 = 1$

D.2. **Further Results for Optimizing Performance.** The second desktop computer ran further tests on locating more parameters that improve the convergence speed and accuracy. Some results are highlighted below in Figs.5-7.

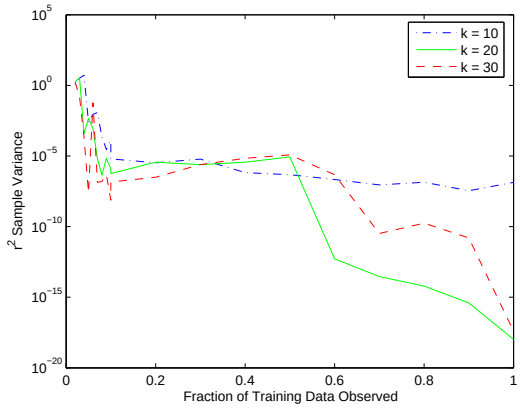

FIGURE 4. Sample variance of prediction error for moderate noise; $\sigma^2 = 0.001$

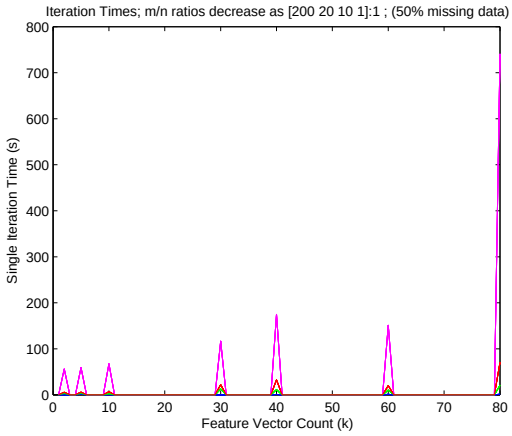

FIGURE 5. Time required for an iteration loop; i.e., time until convergence for a given set of parameters. The plots are, in decreasing order, m/n ratios of 200:1, 20:1, 10;1, and 1:1, which straddles one of our empirically found optimum of 20:1. Please note that the apparently short times, near small k, are due to the algorithm terminating early from very high r^2.

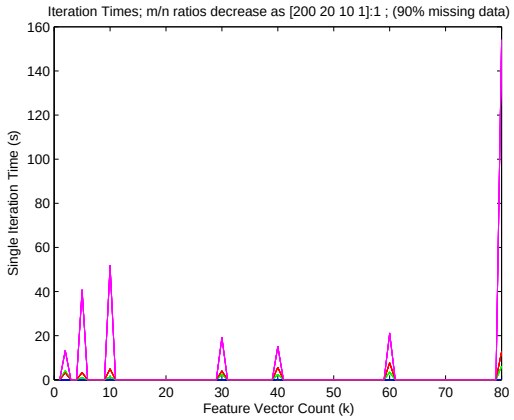

FIGURE 6. Same as Fig.5, but showing decrease in computation time as k increases; for larger m/n values, this increase in performance becomes very small past our determined threshold of $m/n \geq 20:1$

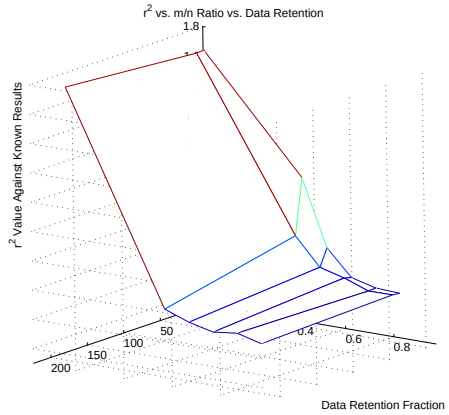

FIGURE 7. Prediction accuracy vs. data retention and m/n ratio. Note the profile shows increasing accuracy (decreasing r^2) as m/n and percent data retention both increase