# Checkmate: A Matrix Completion Approach

Andrew Kessler, Felix Huang, *Department of Mathematics (MIT), Department of Electrical Engineering and Computer Science (MIT)*

***Abstract*—In this paper, we apply matrix completion methods to data from Checkmate, an application that produces compatibility scores between pairs of users. Using techniques like nuclear norm minimization, singular value thresholding, and alternating least squares minimization, we were able to recover scores between unseen pairs of users.**

***Index Terms*—Matrix Completion, Convex Optimization, Low-Rank Matrices, Singular Value Decomposition, Nuclear Norm Minimization, Function Approximation.**

## I. Introduction

### *Matrix Completion*

In many data science scenarios, a practitioner might want to recover unseen information from a larger corpus of tabular data. One may frame this problem as a task in inferring the missing entries of a partially observed matrix, which is termed **matrix completion** [1]. Namely, if given a matrix $\mathbf{M}$ with a set of missing entries $I = \{(i', j'), (i'', j''), ...\}$, the goal of matrix completion is to learn these missing entries $(\mathbf{M}_{(i,j)})_{(i,j)\in U}$ given the truly observed entries $U$. Among applications in system identification, social networks recovery, and collaborative filtering [2], perhaps the most famous instance of matrix completion lies in the *Netflix Problem*, which involves predicting user preferences for media [3]. Here users are represented as rows in a data matrix, with media like TV shows and movies as columns. Each entry, corresponding to a specific user-media-element pair, represents that particular user's rating of the corresponding media element. But naturally, most Netflix users haven't rated (or even seen) every single movie on the platform, so there are missing entries in this matrix construction. By performing matrix completion to fill in these entries, then, Netflix could identify which unseen movies a particular user might rate highly, and thus provide media recommendations to their streaming customers. This is one such example of matrix completion.

But how does one impute these missing entries, especially given the observed entries of a data matrix? A cursory solution might be to select numbers at random–i.e. if $\mathbf{M} \in \mathbb{R}^{\mathbf{mxn}}$, we might sample numbers uniformly along the real line, or, to encourage sparsity, sample from a distribution with high density about zero. A slightly better approach might be to fill in with the average of observed entries across some axis (row-wise or column-wise). In fact, this is the default method for tabular data access in the popular Python data analysis package *Pandas* [4]. In the *Netflix Problem*, for example, one might do column-wise averaging, under the assumption that, without any prior information on a given user, a good estimate for their rating of a movie would be the average rating of that movie, across all users. But is there a smarter way to do this? How can we best exploit the structure of the partially observed matrix and the relationship between entries?

This paper was produced for our final project in Professor Steven's Johnson class 18.065: Matrix Methods in Data Analysis, Signal Processing, and Machine Learning

### *Low Rank Assumptions*

The standard procedure in matrix completion is assume a low-rank structure of the underlying matrix [1], [5]. This allows us to leverage our knowledge about the observed portions of columns (or rows) to filling in entries in dependent columns. For mathematical consistency, we provide the following the definitions. The *rank* of a matrix $\mathbf{M} \in \mathbb{R}^{\mathbf{mxn}}$ refers to the dimensionality of the vector space spanned by its columns (or rows) $dim(C(\mathbf{M})) \leq \min \mathbf{m}, \mathbf{n}$, but for our purposes it's more useful to think of the *rank* as the number of nonzero singular values in the singular value decomposition of $\mathbf{M}$.
The *Singular Value Decomposition* of $\mathbf{M}$ is given by:

$$\mathbf{M} = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T \tag{1}$$

The $\sigma_k$s refer to the singular values of $\mathbf{M}$, which scale the rank one matrices generated by the outer product of the left singular vectors $u_k$s with the right singular vectors $v_k$s.

Low-rank matrix completion makes sense when the size of the data is much greater than the number of "factors" which capture its underlying distribution, or the geometry of its feature space. For example, in the *Netflix Problem* a low-rank assumption is sensible as we assume that only a small set of factors contribute to a user's preferences in media consumption [3]. Matrix completion can then be formulated as:

$$\min_X \quad \text{rank}(X) \tag{2}$$

$$\text{subject to} \quad X_{ij} = \mathbf{M_{ij}} \;\; \forall i, j \in U \tag{3}$$

In other words, we're searching the space of matrices for one which, while matching the entries we've observed, has minimal rank. But this *rank* function takes integer values and is strongly discontinuous; solving matrix completion with this rank minimization is NP-hard [6]. For tractable solutions, low-rank matrix completion requires a convex relaxation.

### *Our Problem*

First, we describe our prediction problem within the **Checkmate** application, and its relation to matrix completion.

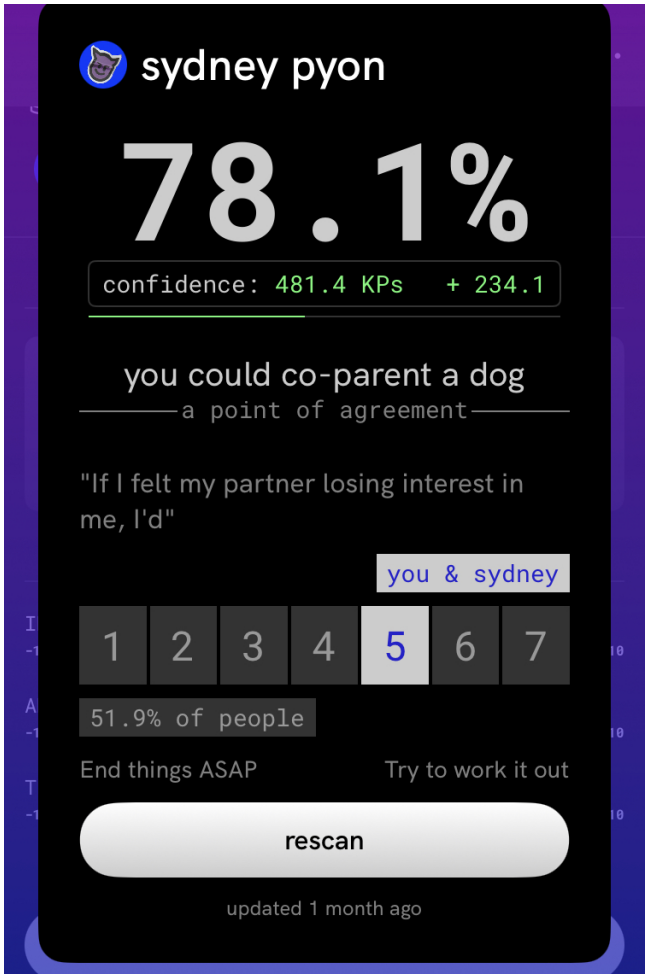



Fig. 1. Sample scan on **Checkmate** between author Andrew Kessler and friend Sydney Pyon, accompanied by a sample question on the app.

**Checkmate** is a mobile app in which users first answer questions concerning their values and interests, and then "scan" one another to receive a "compatibility" score (a percentile between 0 and 100) that purportedly represents the quality of friendship that could arise between them [7].

We want to predict novel compatibility between users on the app who have not yet scanned one another, based on their known compatibilities with common third-party users. This has a convenient matrix completion formulation: for a set of users $U$, we can construct a symmetric matrix $\mathbf{M} = \mathbf{M}^T$, in which $\forall i, j \in U$, $\mathbf{M}_{ij} = \mathbf{M}_{ji}$ is the compatibility score between user $i$ and user $j$. Based on the definition of compatibility, all entries of $\mathbf{M}$ are on $[0, 1]$, and entries on the diagonal are 1, as a user is wholly compatible with him/herself. Every user has not scanned every other user, so this matrix is incomplete. Predicting unscanned compatibilities, then, is equivalent to filling in these empty entries, or completing the matrix. Our low-rank approximation is also valid here: it's reasonable to assume that there exist a small number of friend groups, or personality types, which influence compatibility [8].

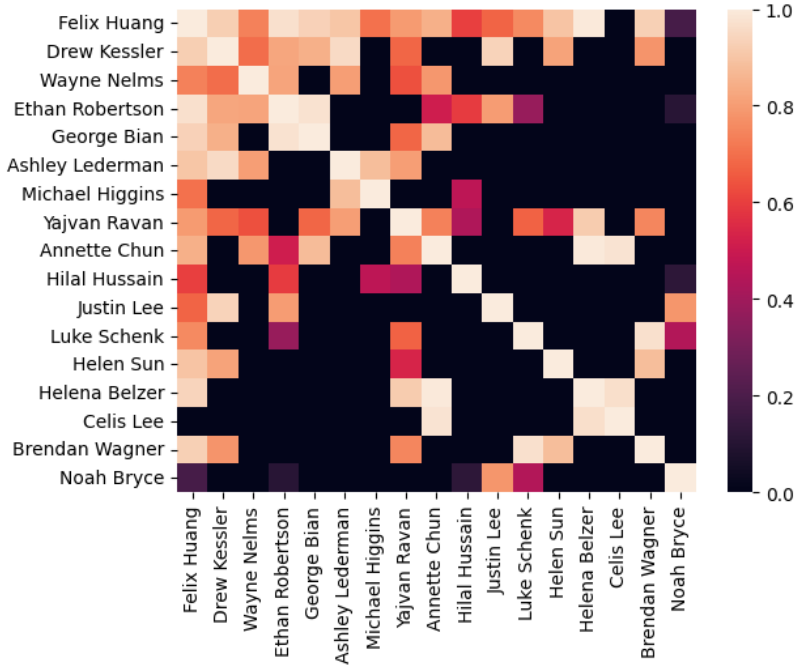



Fig. 2. Heatmap of our data on **Checkmate** among the seventeen users in the sample. The black squares corrrespond to unobserved compatibility scores, or empty entries in the matrix.

### *Data Collection*

We sampled 52 scans from an enclosed set of 17 users. This means that we're observing $52 * 2 + 17 = 121$ entries in our realization of $\mathbf{M}$, because of the symmetry of compatibility as well as knowing that scans along the diagonal are all 1. These were all collected simultaneously on April 8, 2023, as users can repeat scans (after answering new questions) after a cooloff period of twenty four hours. The proportion of unobserved entries in $\mathbf{M}$, $\frac{289-121}{289} \approx 0.62$, is in-line with both empirical suggestions and informational-theoretical necessities [5], [9]. Most importantly, we have at least two observed entries in every column and row of $\mathbf{M}$–were this not the case, matrix completion would be infeasible because the complete lack of information in some direction would lead to non-uniqueness in a solution's singular vectors [10]. Having satisfied these constraints, we aim to perform low rank matrix completion on this $\mathbf{Checkmate}$ data.

## II. Methods

### *Nuclear Norm Minimzation*

We know that low-rank matrix completion, in its canonical form as described by the objective and constraint in (2) and (3), is NP-hard, so we must relax the problem into a more tractable format. It turns out that a good heuristic for the rank of a matrix, when normalizing the singular vectors, is the *nuclear norm* of a matrix, given by the sum of its singular values:

$$\|\mathbf{M}\|_* = \sum_{\mathbf{k=1}}^{\mathbf{r}} \sigma_{\mathbf{k}}(\mathbf{M}) \tag{4}$$

Fazel showed that *nuclear norm* minimization was a good convex relaxation of low rank matrix completion [11]. Specifically, he proposed the following formulation instead of the rank minimization outlined in (2) and (3).

$$\min_{X} \quad \|\mathbf{X}\|_* \tag{5}$$
$$\text{subject to} \quad X_{ij} = \mathbf{M_{ij}} \ \ \forall i,j \in U \tag{6}$$

Candes and Recht then transformed this approach of *nuclear norm* minimization by refromulating it into an equivalent semidefinite program [10]:

$$\min_{W_1,W_2} \quad \text{trace}(W_1) + \text{trace}(W_2) \tag{7}$$
$$\text{subject to} \quad X_{ij} = \mathbf{M}_{ij} \ \ \forall i,j \in U \tag{8}$$
$$\begin{bmatrix} W_1 & X \\ X^T & W_2 \end{bmatrix} \succeq 0 \tag{9}$$

The use of semidefinite programming in this procedure for *nuclear norm* minimization drastically improves the feasibility of low-rank matrix completion: when $\mathbf{M}$ is of shape $(m,n)$, it can be shown to run in $O((\max m,n)^4)$ time [12]. Additionally, probabilistic analysis confirms the approximation of rank minimization by *nuclear norm* minimization [10], [12].

*Singular Value Thresholding*

A sister approach to *nuclear norm* minimization is called singular value thresholding, a process which iteratively thresholds the singular values of $\mathbf{M}$ to achieve a low rank solution. Borrowing notation from Cai, Candes, and Shen [12], we can define the operator $P_U : \mathbb{R}^{mxn} \to \mathbb{R}^{mxn}$ to be the othorgonal projector where:

$$P_U(X)_{i,j} = \begin{cases} \mathbf{M}_{ij}, & \text{if } (i,j) \in U \\ 0, & \text{otherwise.} \end{cases}$$

An equivalent formulation of the *nuclear norm* minimization objective (5) and (6) then becomes

$$\min_{X} \quad \|\mathbf{X}\|_* \tag{10}$$
$$\text{subject to} \quad P_U(X) = P_U(\mathbf{M}) \tag{11}$$

The general approach to singular value thresholding entails an inductive process. To threshold the singular values at level $\tau$ and a sequence of step sizes $(\delta_k)_{k\geq 1}$, starting with a zero matrix $Y^0 = 0$ until stopping we update

$$X^k = \text{threshold}(Y^{k-1}, \tau) \tag{12}$$
$$Y^k = Y^{k-1} + \delta_k P_U(\mathbf{M} - X^k) \tag{13}$$

This procedure converges to a matrix which well approximates the solution to the optimization problem defined by (10) and (11). Importantly, at each step we only compute one singular value decomposition in the threshold function, which always outputs a low rank matrix (in-line with the overall objective of low rank matrix completion) by the sparsity of each $Y^k$ induced by the operator $P_U$ [2], [12].

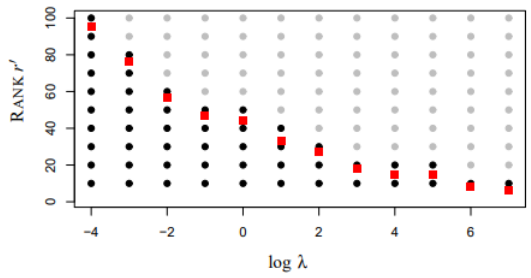



Fig. 3. From [13], a comparison of the iterative singular value thresholding (red) with other standard methods of low-rank matrix completion (black, gray). Here $\lambda$ is our $\tau$, as in the threshold. This validates the procedure, as more stringent thresholding (via the variation in the parameter space) lower and lower rank outcomes.

Specifically, our implementation of singular value thresholding will use the ratios of successive Frobenius norms. The *Frobenius norm* of a matrix $\mathbf{M}$ is given by:

$$\|\mathbf{M}\|_F = (\sum_{k=1}^{r} (\sigma_k(\mathbf{M}))^2)^{0.5} \tag{14}$$

To perform the thresholding at level $\tau$ on $X$, we transform the singular value decomposition of $X = U\Sigma V^T$ into its thresholded version, keeping the left and right singular vectors but substituting $\Sigma$ for $\Sigma' = \text{diag}[(\max(\sigma_1 - \tau), 0), ..., (\max(\sigma_r - \tau), 0))]$. So $S_\tau(X) = \text{threshold}(X,\tau) = U\Sigma' V^T$. This gives the following implementation:

**Algorithm 1** Iterative Singular Value Thresholding with Frobenius stopping condition.

INITIALIZE $Y^0 = 0$

**Repeat**
a. $X^k \leftarrow S_\tau(Y^{k-1})$
b. $Y^k \leftarrow Y^{k-1} + \delta_k P_U(\mathbf{M} - X^k)$
c. If $\frac{\|Y^k - Y^{k-1}\|_F}{\|Y^{k-1}\|_F} < \epsilon$, EXIT

**Return** $Y^k$.

This process is dependent on our choice of convergence ratio $\epsilon$ as well as the structure of the sequence of step sizes $\delta_k$, which we chose to be exponentially decreasing. One desirable feature of Algorithm 1 is that, in each step, we update the missing entries of the desired matrix $\mathbf{M}$ with our current estimate from $Y^k$. Mazumder, Hastie, and Tibshirani showed that this converges under weak assumptions with a favorable worst-case rate [13].

*Alternating Least Squares Minimization*

The final procedure we consider for our low rank matrix completion task is that of alternating least squares minimiza-

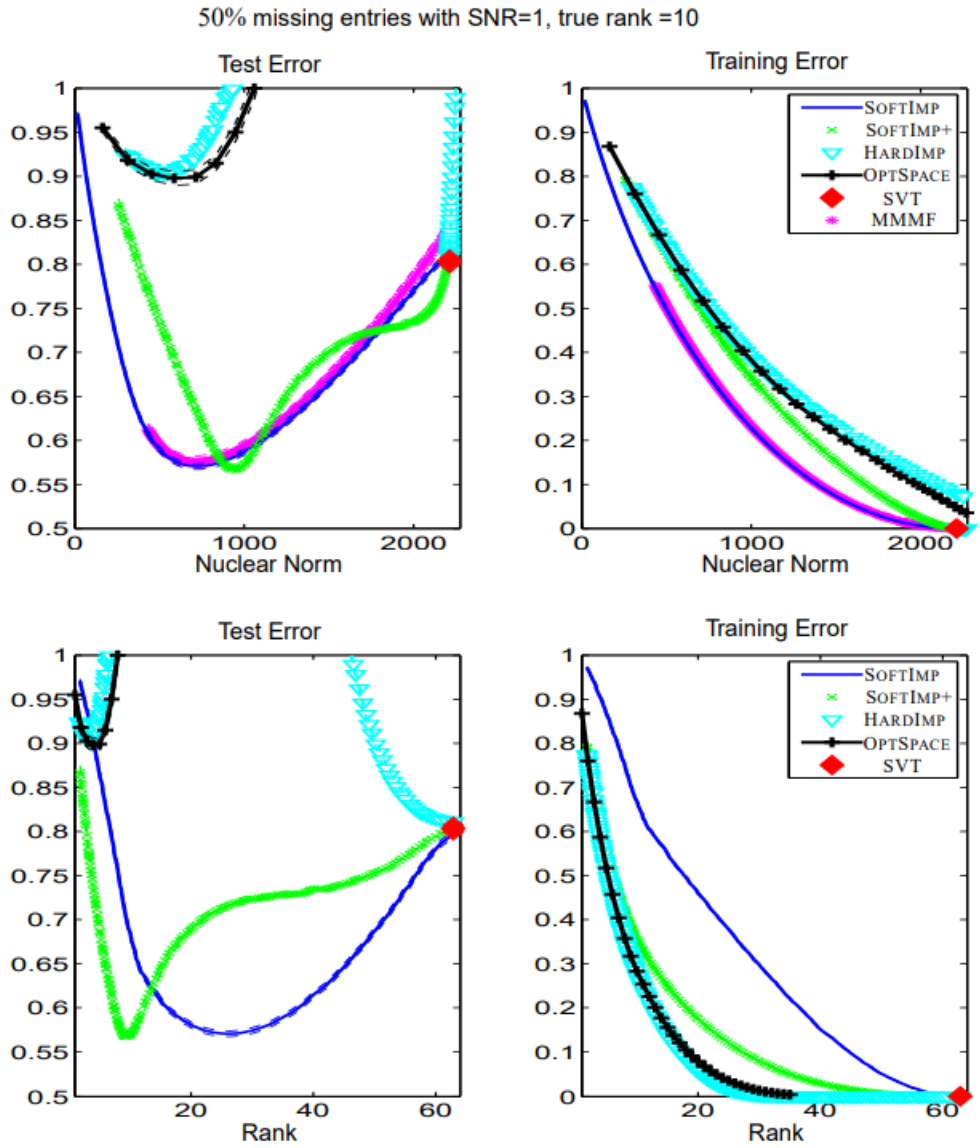



Fig. 4. From [13], a comparison of the iterative singular value thresholding (SoftImp) with other standard methods of low-rank matrix completion. Here 50 percent of the entries are missing, and there is some random Gaussian noise. The figure indicates that the algorithm generalizes well and outputs low rank, low *nuclear norm* matrices whose filled-in entries are quite close to their true values.

tion. Because the *Frobenius norm* is essentially the L2-norm of the singular values (and the *nuclear norm* is L1), yet another formulation of the rank minimization program outlined in (2) and (3) involves assuming a bilinear form $X = AB^T$ for $A \in \mathbb{R}^{mxk}, B \in \mathbb{R}^{nxk}$, where we assume $M$ to be a small rank $k$.

$$\min_{A,B} \qquad \|P_U(AB^T) - P_U(\mathbf{M})\|_F \qquad (15)$$

By introducing some randomness and alternating between finding the best $A$ and the $B$, we can efficiently reduce this nonconvex objective. This gives an alternating least squares process, proposed by Jain, Netrapalli, and Sanghavi [14].

**Algorithm 2** Alternating Least Squares Minimization

1. PARTITION the observed set of entries $U$ into $2N+1$ subsets $U_0, ..., U_{2N}$ by sampling with replacement

2. SET $\hat{A}^0$ to the top-$k$ left singular vectors of $\frac{1}{p}P_{U_0}(\mathbf{M})$.

3. SET all elements of $\hat{A}^0$ with magnitude greater than $\frac{2\mu\sqrt{k}}{\sqrt{n}}$ to zero and orthonormalize the columns of $\hat{A}^0$.

4. **for** $t = 0, ..., N-1$ **do**
5. a) $\hat{B}^{t+1} \leftarrow \text{argmin}_{B\in\mathbb{R}^{nxk}} \|P_{U_{t+1}}(\hat{A}^t B^T - \mathbf{M})\|_F^2$
6. b) $\hat{A}^{t+1} \leftarrow \text{argmin}_{A\in\mathbb{R}^{mxk}} \|P_{U_{N+t+1}}(A(\hat{B}^{t+1})^T - \mathbf{M})\|_F^2$

7. RETURN $X = \hat{A}^N(\hat{B}^N)^T$.

This algorithm has hyperparameters $p$ and $T$, which regulate the stopping condition in the approach. $\mu$ refers to the incoherence parameter of $\mathbf{M}$, which allows us to control the sparsity of solutions. The steps in lines 5 and 6 can be performed by a fast least squares projection, and with regularity conditions on $\mathbf{M}$, alternating least squares minimization has asymptotic runtime proportional to $O(|U|k^2)$ [1], [14].

We selected these three methods–*nuclear norm* minimization, singular value thresholding, and alternating least squares minimization–as efficient convex reductions for low rank matrix completion, on the basis of their theoretical guarantees as well as suitability with our data. To translate these approaches into working code, we used Python and leveraged the convex solver packages *CVXOPT* and *FancyImpute* [15], [16].

## III. Results

*Validation*

First, we need to validate our implementation of these methods in code. For simulation we used the underlying model $Z_{mxn} = A_{mxr}B^T_{nxr} + \epsilon_{mxn}$, with $A, B$'s elements sampled from a standard normal, and $\epsilon$'s elements from smaller noise generated by a gaussian with mean zero and variance $0.1$. The missing entry set $E$ was random with $p$ percent of the available indices. To evaluate the goodness of fit for the completed matrices generated by our model, we used the following statistics, consistent with [13].

$$\text{Test Error} = \frac{\|P_U^{\perp}(AB^T - \hat{Z})\|_F^2}{\|P_U^{\perp}(AB^T)\|_F^2} \tag{16}$$

$$\text{Training Error} = \frac{\|P_U(Z - \hat{Z})\|_F^2}{\|P_U(Z)\|_F^2} \tag{17}$$

Armed with these definitions, we followed the following validation procedure:

**Algorithm 3** Validation of Various Low Rank Matrix Completion Methods

INPUT $(m, n)$: shape of random matrix $Z$, $r$: rank of matrices $A, B$, $p$: percent of observable indices, $I$: total number of samples for validation

2. **for** $i = 1, ..., I$, **do**

a) SAMPLE $A \in \mathbb{R}^{mxr}$ and $B \in \mathbb{R}^{nxr}$ from a standard normal Gaussian, $\epsilon \in \mathbb{R}^{mxn}$ from a Gaussian with mean zero and variance $0.1$, and the set of observed indices $E$ from $[m] \cdot [n]$ with $p$ percent of available indices.

b) SET $Z = AB^T + \epsilon$.

c) EVALUATE $\hat{Z}^i_{NNM} = \text{NNM}(P_U(Z))$, $\hat{Z}^i_{SVT} = \text{SVT}(P_U(Z), \tau = 0.1)$, and $\hat{Z}^i_{ALT} = \text{ALT}(P_U(Z))$

d) EVALUATE test error $E_{Test}$ and $E_{Train}$ on all three completed matrix estimates.

3. **Return** the sequences of tuples $(E^i_{Train,NNM}, E^i_{Train,SVT}, E^i_{Train,ALT})_{1 \le i \le I}$ and $(E^i_{Test,NNM}, E^i_{Test,SVT}, E^i_{Test,ALT})_{1 \le i \le I}$ for analysis.

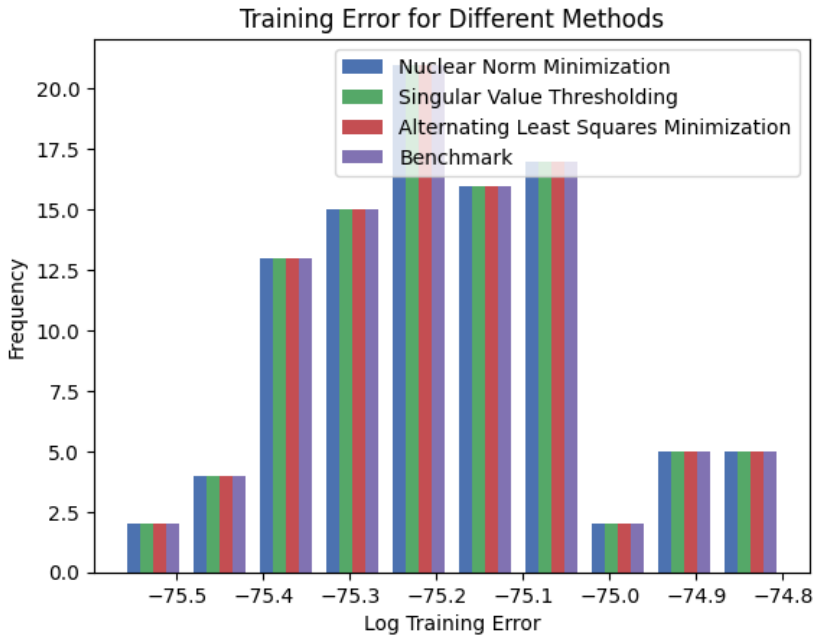
So with $m, n = 17$, $p = 0.5$, $r = 5$, $I = 100$ samples of $Z$, in testing the three methods of matrix completion against a simple benchmark of filling with the columnwise averages, all three outperformed. *Nuclear norm* minimization had an average test error of $0.428$, singular value thersholding had $0.527$, and finally alternating least squares minimization had a test error of $0.6$. Our benchmark, however, produced an error of $1.18$. As a robustness check, all methods (including the benchmark), produced effectively zero training error, which we expect because the operator $P_U$ projects matrices onto the set of observed entries. Any algorithm that keeps the observed entries the same in matrix completion (as is the case with all of our methods) will give a training error that's effectively zero, so the validation data matches this essential feature. See figures 5a and 5b for additional analysis.

*Application on* **Checkmate** *Data*

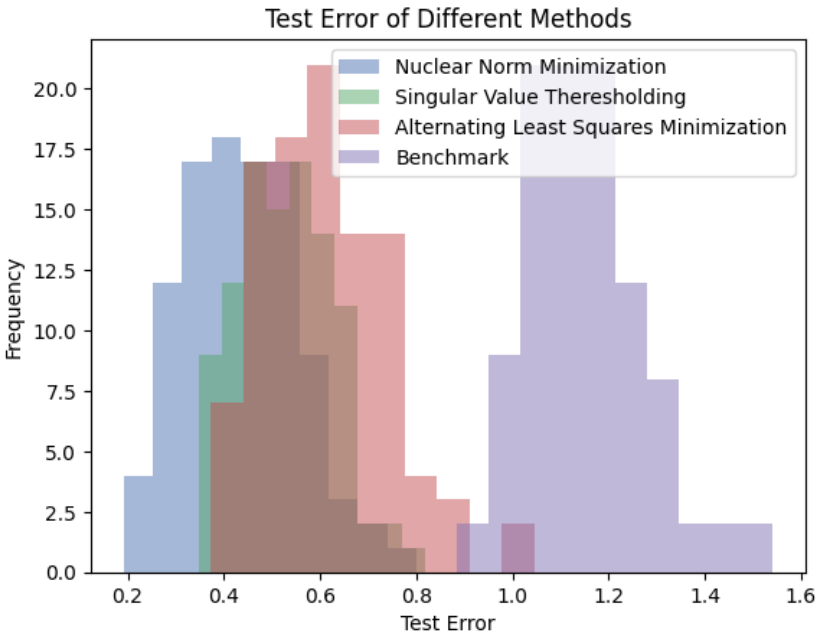
Now that we've validated our three methods for low rank matrix completion, we apply them to our **Checkmate** Data matrix $\mathbf{M}$ with approximately $38$ percent entry observation. Importantly, without imposing the symmetry constraint that's present in the **Checkmate** context (the compatibility score between user $i$ and user $j$ is the same as the score between user $j$ and user $i$), all three methods gave symmetric matrices as output. Column-wise averaging does not recover this feature of $\mathbf{M}$. The associated heatmaps for our estimates $\hat{M}_{NNM}$, $\hat{M}_{SVT}$, $\hat{M}_{ALSM}$ from *nuclear norm* minimization, singular value threshold (with a Frobenius stopping condition), and alternating least squares minimization, respectively, are depicted in figures 6a, 6b, and 6c.

As in our validation procedure, *nuclear norm* minimization took the longest at $2.526$ seconds on Google Colab, with singular value thresholding and alternating least squares minimization taking $1.934$ and $2.049$ seconds, respectively. As alluded to in part IV, testing on more larger partially observed matrices needs to be done to make any claim as to the practical runtimes of these methods.

While these results indicate that the completed matrices generated by *nuclear norm* minimization, singular value thresholding, and alternating least squares minimization are reasonable estimates for the true compatibility matrix $\mathbf{M}$ that's masked by the observation set $U$, we don't know yet exactly *how* good they are. We executed the validation procedure outlined in Algorithm 3, which showed that these processes, when applied randomly generated matrices $Z$ (with additional noise $\epsilon$), dramatically outperforms a column-wise averaging, and the estimated matrices $\hat{Z}$ are "close" to $Z$ under the test error metric that uses the Frobenius norm. But this doesn't necessarily imply that for our specific **Checkmate** data matrix, the output estimates $\hat{\mathbf{M}}$ match the true entries that are first unobserved in $\mathbf{M}$. So the following day we actually acquired the compatibility scores of the unscanned pairs of users in $\mathbf{M}$. We did this prior to the next set of daily questions arriving, so as to ensure that the already-observed compatibility scores of $\mathbf{M}$ would not be perturbed. This fully observed realization of $\mathbf{M}$ is depicted in figure 7a.




(a)




(b)

Fig. 5. Validation of our three methods in recovering a randomly produced matrix of rank 5 with additional noise (a) Histogram of training errors as defined in (17). (b) Histogram of test errors as defined in (16). The benchmark procedure for matrix completion was filling in with a simple columnwise average. As expected, the training error for all methods was zero because the observed entries in the partially observed matrices $Z$ were the same as in the estimates $\hat{Z}$. *Nuclear norm* minimization seemed to perform the best in completing the underlying matrix (without noise), which is likely because the semidefinite program is most general; it requires no hyperparameters, but with singular value thresholding we send all singular values less than $\tau = 0.1$ to 0, and alternating least squares minimization first has to estimate the underlying rank so it can represent its iterative estimates in bilinear form. Notably, however, *nuclear norm* minimization took the longest on Google Colab, averaging 3.21 seconds per sample, 0.64 seconds greater than its next slowest counterpart.

To analyze the difference between our method-generated estimates and the fully observed matrix, we'll use a root mean squared error (RMSE) statistic [2], [17]. By averaging over the squared deviation in our estimates and the newly-observed entries $E = [m] \cdot [n] \setminus U$ of $\mathbf{M}$, we get:

$$\text{RMSE}(\hat{M}, \mathbf{M}) = \sqrt{\frac{\sum_{(i,j)\in E}(\hat{M}_{ij} - \mathbf{M}_{ij})}{|E|}} \tag{18}$$

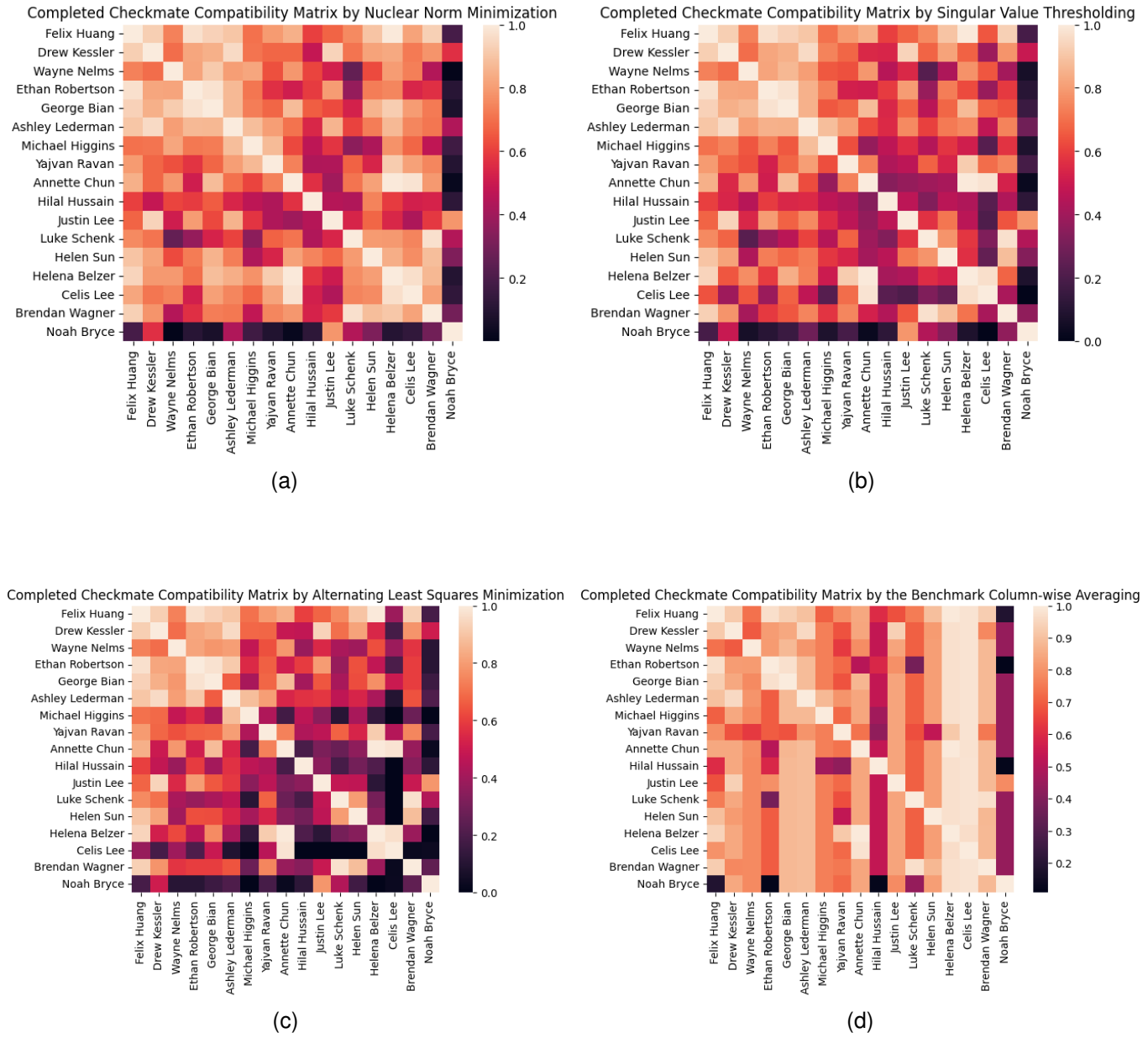



Fig. 6. Completed Matrices based on our three methods. (a) Heatmap of compatibility scores predicted by *Nuclear norm* minimization, (b) Heatmap of compatibility scores predicted by singular value thresholding with a Frobenius stopping condition, (c) Heatmap of compatibility scores predicted by alternating least squares minimization with prior rank estimation, and (d) Heatmap of compatibility scores predicted by our benchmark method, column-wise averaging. Observe the symmetry in subfigures (a)-(c); this is as desired with the symmetry in compatibility scores. Also notice the relative sparsity of alternating least squares minimization when compared to our two other methods–this difference may be due to optimizing over a Frobenius norm objective and the incoherence parameter $\mu$ which promotes sparsity. It's clear that the three methods outperform our benchmark by leveraging more global information about the observed entries of $\mathbf{M}$ in the set $U$, while the benchmark just employs local data in each column.

Our methods gave $\text{RMSE}(\hat{M}_{NNM}, \mathbf{M}) = 0.121$, $\text{RMSE}(\hat{M}_{SVT}, \mathbf{M}) = 0.036$, $\text{RMSE}(\hat{M}_{ALSM}, \mathbf{M}) = 0.122$, compared to an RMSE value of $0.466$ for the benchmarked estimate generated by columnwise averaging. This indicates that all three of our methods of low rank matrix completion are outperforming the naive benchmark procedure for this **Checkmate** context, and that for this realization of the data matrix $\mathbf{M}$, singular value thresholding with a Frobenius stopping condition performs best. Visually, the closeness of the heatmaps for the true matrix in 7a and the singular value thresholding estimate in 7b supports this hypothesis. Nonetheless, more testing needs to be done to make a conclusive decision about the superiority of any of the three methods in general.

## IV. Conclusion

We first saw matrix completion, a data imputation process in which one is tasked with systematically filling in the unobserved entries of a partially observed matrix, through the context of the *Netflix Problem* [3]. From Netflix's perspective,

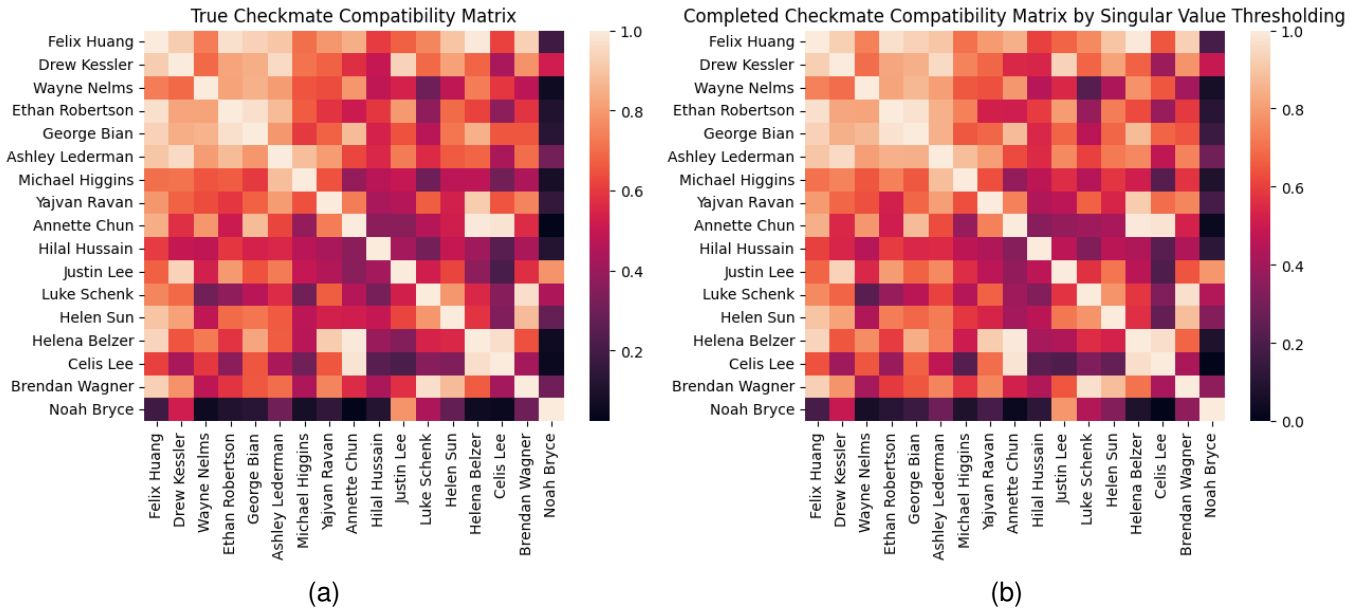



Fig. 7. Comparison of the fully observed compatibility matrix to our closest estimate, generated by singular value thresholding. (a) Heatmap of true compatibility scores, sourced from the **Checkmate** app, (b) Histogram of the compatibility scores generated by singular value thresholding, the closest estimate among the three methods we considered. This method achieved an RMSE value of $0.036$ with respect to the fully observed matrix $\mathbf{M}$ depicted in (a), which is superior to the RMSE value of the benchmark estimate, $0.466$, generated by column-wise averaging.

it has access to an partially-filled matrix of users' media ratings. This tabular structure encodes a user $i$'s rating of Netflix's $j$th movie, but because not every user has seen every movie on Netflix, the matrix is incomplete. To predict a user's rating on an unseen movie, then, is to fill in an empty element in this data matrix, which naturally lends itself to matrix completion. For our application, we used data from the **Checkmate** mobile app, in which users scan one another to gain compatibility scores corresponding to their fitness for friendship [7]. We sampled a set of these scans and generated a partially-observed symmetric matrix in which the $(i, j)$th element corresponds to the compatibility between user $i$ and user $j$. As in the NETFLIX PROBLEM, not every user had scanned every other user, so to predict unscanned compatibilities we turned to matrix completion.

By reasoning that there were only a few fundamental "factors" associated with compatibility, we subscribed to a low rank assumption commonly used in the field [1], [5]. But the objective of minimizing rank subject to constraining the observed entries in the matrix, outlined in $(2)$ and $(3)$, involved searching through an exponentially-large search space, so we turned to convex relaxations of this program. By observing the close relationship between the *rank* of a matrix (the number of nonzero singular values), the *nuclear norm* (the sum of singular values), and the *Frobenius norm* (the square root of the sum of squared singular values), we decided on using the techniques of nuclear norm minimization, singular value thresholding, and alternating least squares minimization.

Nuclear norm minimization, a procedure first proposed by Candes and Recht [10], involves proxying the rank of a matrix by its nuclear norm (see $(5) - (9)$), and has been shown to efficiently converge to the true underlying matrix with high probability. An iterative process for this optimization is known as singular value thresholding, which at every stage updates its estimate to become closer to the observed entries of the underlying matrix $\mathbf{M}$, and then performs a modified singular value decomposition that sends small singular values to zero in order to lower the rank of its estimate. Our implementation of singular value thresholding used a Frobenius stopping condition [13]. Essentially, we stopped iterating when the ratio of the Frobenius norms of our estimate didn't change all that much. Finally, we employed an alternating least squares minimization that assumes a bilinear structure of $M = AB^T$ and repeatedly optimizes $A$ and $B$ by least squares. This procedure had nice efficiency and sparsity controls [14].

Before applying these three methods on our **Checkmate** data, we validated them by generating random 17 by 17 matrices of rank 5, randomly selecting some elements to leave unobserved, and seeing how well each method could recover these entries. When compared to a benchmark matrix completion algorithm that just fills in by executing a column-wise average, nuclear norm minimization, singular value thresholding, and alternating least squares minimization significantly outperformed under the test error metric $(16)$ (see figure 5b).

Applying the three methods to our **Checkmate** data, each estimate recovered the underlying symmetry of compatibility scores, which wasn't the case for the benchmark column-wise averaging. We then evaluated each method's estimate against the previously unobserved compatibility scores, and

via a root-mean-squared-error metric they all bested the same benchmark. Ultimately, for this instance of **Checkmate** data, singular value thresholding proved the best method with an RMSE of $0.036$.

*Next Steps*

We only had one sample of **Checkmate** data in this experiment, which is surely insufficient to conclude the superiority of any one of our matrix completion methods. More testing (with a cross-validation for optimizing each method's hyperparameters) with data matrices of different shapes needs to be done before any statistical analysis.

Outside of the raw compatibility score, there are other **Checkmate** features which could assist in compatibility prediction. The most glaringly useful are the numerical answers to the questions that identify personality traits (see figure 1) and are the underlying components in computing compatibility scores. In the future perhaps a sample of these answers could be ascribed to every user, in addition to the partially-observed compatibility data that's accessible, to give more informed predictions.

Finally, we could consider other data imputation methods outside of the three employed in this article. One unconventional tool that's commonly adopted in practice is a nearest-neighbors-based approach, which fills in missing entries via some function of its nearest observed elements [18]. Overall, however, we hope to have convinced the reader of the feasibility of applying matrix completion techniques to this **Checkmate** problem, and we're excited to continue with further experimentation.

## ACKNOWLEDGMENTS

Our thanks to Arun Kirk for entertaining our difficulties with the project.

## V. REFERENCES SECTION

## REFERENCES

[1] M. Laurent, "Matrix Completion Problems," Encyclopedia of Optimization, pp. 1967–1975, 2008, doi: https://doi.org/10.1007/978-0-387-74759-0-355.

[2] E. J. Candes and Y. Plan, "Matrix Completion With Noise," Proceedings of the IEEE, vol. 98, no. 6, pp. 925–936, Jun. 2010, doi: https://doi.org/10.1109/jproc.2009.2035722.

[3] ACM SIGKDD, Netflix, Proceedings of KDD Cup and Workshop (2007). Proceedings available online at http://www.cs.uic.edu/ liub/KDD-cup-2007/proceedings.html.

[4] W. McKinney and others, "Data structures for statistical computing in python," in Proceedings of the 9th Python in Science Conference, 2010, vol. 445, pp. 51–56.

[5] Z. Xu, "The minimal measurement number for low-rank matrix recovery," Applied and Computational Harmonic Analysis, vol. 44, no. 2, pp. 497–508, Mar. 2018, doi: https://doi.org/10.1016/j.acha.2017.01.005.

[6] C. Johnson, Ed., Matrix Theory and Applications. Providence, Rhode Island: American Mathematical Society, 1990. doi: https://doi.org/10.1090/psapm/040.

[7] Checkmate. Marriage Pact. https://apps.apple.com/us/app/checkmate-scan-your-friends/id6443729738

[8] K. N. Kelley, "Using the Myers-Briggs Type Indicator," PsycCRITIQUES, vol. 50, no. 33, 2005, doi: https://doi.org/10.1037/05159912.

[9] E. J. Candès and T. Tao, "The Power of Convex Relaxation: Near-Optimal Matrix Completion," IEEE Transactions on Information Theory, vol. 56, no. 5, pp. 2053–2080, May 2010, doi: https://doi.org/10.1109/tit.2010.2044061.

[10] E. J. Candès and B. Recht, "Exact Matrix Completion via Convex Optimization," Foundations of Computational Mathematics, vol. 9, no. 6, pp. 717–772, Apr. 2009, doi: https://doi.org/10.1007/s10208-009-9045-5.

[11] M. Fazel, "Matrix Rank Minimization with Applications," Stanford University, 2002. Accessed: May 14, 2023. [Online]. Available: https://faculty.washington.edu/mfazel/thesis-final.pdf

[12] J.-F. Cai, E. J. Candès, and Z. Shen, "A Singular Value Thresholding Algorithm for Matrix Completion," SIAM Journal on Optimization, vol. 20, no. 4, pp. 1956–1982, Jan. 2010, doi: https://doi.org/10.1137/080738970.

[13] R. Mazumder, T. Hastie, and R. Tibshirani, "Spectral Regularization Algorithms for Learning Large Incomplete Matrices.," Mar. 2010.

[14] "Low-rank matrix completion using alternating minimization — Proceedings of the forty-fifth annual ACM symposium on Theory of Computing," ACM Conferences, 2013. https://doi.org/10.1145

[15] M. S. Andersen, J. Dahl, and L. Vandenberghe. CVXOPT: A Python package for convex optimization, version 1.1.5. Available at abel.ee.ucla.edu/cvxopt, 2012.

[16] A. Rubinsteyn and S. Feldman, fancyimpute: An Imputation Library for Python. 2016. [Online]. Available: https://github.com/iskandr/fancyimpute

[17] Z. Huo, J. Liu, and H. Huang, "Optimal Discrete Matrix Completion," Proceedings of the AAAI Conference on Artificial Intelligence, vol. 30, no. 1, Feb. 2016, doi: https://doi.org/10.1609/aaai.v30i1.10300.

[18] L. Beretta and A. Santaniello, "Nearest neighbor imputation algorithms: a critical evaluation," BMC Medical Informatics and Decision Making, vol. 16, no. S3, Jul. 2016, doi: https://doi.org/10.1186/s12911-016-0318-z.